P.S.I.

Le BASIC et ses Fichiers

méthodes pratiques

Jacques Boisgontier

Editions du P.S.I.

Le BASIC et ses FICHIERS

méthodes pratiques

Dans la même collection

Programmer en BASIC - Michel Plouin
Programmer en L.S.E. — Stéphane Berche et Yves Noyelle
Comment programmer — Jean-Claude Barbance
La découverte de l'Apple II — Frédéric Lévy et Dominique Schraen
La découverte du P.E.T. - Daniel-Jean David
La pratique du P.E.T./C.B.M. - Volume I — Daniel-Jean David
La pratique du TRS-80 - Volume I — Pierre Giraud et Alain Pinaud
La pratique du TRS-80 - Volume II — Pierre Giraud et Alain Pinaud
La pratique du TRS-80 - Volume III — Pierre Giraud et Alain Pinaud

Tous droits de traduction, d'adaptation et de reproduction par tous procédés
réservés pour tous pays

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les «copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective» et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, «toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite» (alinéa 1er de l'article 40).

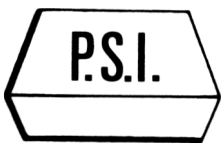
Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

1ère édition - Mai 1980
© *Editions du P.S.I. 9, rue d'Orgemont 77400 LAGNY/MARNE - 1980*
ISBN : 2-86470-010-7

Imprimé en France

Le BASIC et ses FICHIERS méthodes pratiques

par
Jacques Boisgontier



Editions du P.S.I.
1980

AVERTISSEMENT

Cet ouvrage est destiné aux informaticiens individuels qui souhaitent développer sur leur P.S.I. des programmes mettant en oeuvre des *fichiers sur disquette ou sur disque.*

Devant la multiplicité des BASIC et des Systèmes d'Exploitation Disque, il a fallu faire un choix que nous avons voulu le moins restrictif possible. Nous avons retenu le *Basic Microsoft* dans sa dernière version : la version 5. fonctionnant sous *système CP/M.*

Ce livre s'adresse donc, en priorité, aux utilisateurs de *P.S.I. dotés d'un microprocesseur Z-80.* Cependant, *ceci n'est pas totalement restrictif* puisque, par exemple, le BASIC du X1 de la Société Occitane d'Electronique, dont le microprocesseur est un 6800, est, à quelques différences près, celui décrit dans cet ouvrage.

A moment où ce livre est mis sous presse, on peut citer, parmi les P.S.I. directement concernés : TRS 80 (avec quelques restrictions signalées) ; Zenith (Heathkit) Z8, Z89 ; LX500 avec S.E.D. SX/80 ; Occitane X1 ; North Star ; Cromenco ; Altos ; Sanco ; Vector ; Micral 80 ; PCC 2000 ; Kontron PSI 80; ISTC 5000 option CP/M ; DTC Microfile ; CB 7900 ; Industrial Micro System ; ...

Jacques BOISGONTIER *a enseigné l'informatique industrielle, la téléinformatique et le Basic à la Régie Nationale des Usines Renault.*

Actuellement, il se consacre à l'automatisation de bancs d'essais.

SOMMAIRE

PRESENTATION 7

Ière PARTIE : LE BASIC MICROSOFT 9

Editeur de programme 11
Commandes BASIC 13
Variables 15
Expressions et opérateurs 18
Entrées au clavier 20
Tests 21
Boucle FOR 22
Tableaux 25
Données 27
Chaînes de caractères 28
Editions 35
Sous-programmes 40
Branchements multidirections 41
Traitement d'erreurs 42
Fonctions arithmétiques 43
Définition de fonctions 45
Appel de programmes 47
Accès direct en mémoire 48

IIème PARTIE : LES FICHIERS EN BASIC MICROSOFT 55

Chapitre 2-1 - FICHIERS A ACCES DIRECT 57

Chapitre 2-2 - FICHIERS SEQUENTIELS 69

IIIème PARTIE : METHODES PRATIQUES 77

Chapitre 3-1 - QUELQUES METHODES SIMPLES 79

Date fichier 79
Liste triée d'un fichier à accès direct 80
Regroupement par code 83
Allocation de l'espace disque 84
Allocation dynamique 85
Fichier à accès direct virtuel 87
Saisie écran TRS80 89
Saisie écran LX500 91

Chapitre 3-2 - L'ACCES INDEXE 93

Accès indexé avec table en mémoire centrale 94
Sauvegarde sur disque d'une table d'index 96
Index continu en ordre croissant 99
Index 2 niveaux 99
Accès indexé TRS80 100

.../...

SOMMAIRE (suite)

Chapitre 3-3 - LES BASES DE DONNEES 105

Pourquoi les bases de données 106
Gestion des pointeurs 107
Exemple de base 114

ANNEXES

Annexe I Rappels sur BASIC 119
Annexe II Messages d'erreurs 133
Annexe III Table des codes ASCII 137
Annexe IV Liste des programmes 138
Annexe V Index alphabétique 139

PRESENTATION

"Le Basic et ses fichiers" s'adresse, en principe, à des lecteurs déjà familiarisés avec le Basic ; *toutefois, un rappel des notions fondamentales du langage est à la disposition des néophytes en Annexe I.*

Nous pensons que de nombreux adeptes du Basic pourront, également, trouver un intérêt à la lecture de ces rappels.

L'ouvrage se compose de trois parties :

La première associe à la description du langage Basic Microsof 5 des exemples d'utilisation. Pour tirer le meilleur parti de cet ouvrage sur son propre matériel, le lecteur pensera à rapprocher cette description du manuel de référence de son P.S.I. afin de détecter les différences éventuelles.

La deuxième partie s'intéresse aux fichiers et à leurs instructions spécifiques avec, comme en première partie, des exemples spécifiques *de fichiers à accès direct et à accès séquentiel sur disque.*

La troisième partie est entièrement consacrée à l'exposé de méthodes pratiques d'utilisation des fichiers depuis l'accès indexé jusqu'aux bases de données.

Enfin, deux annexes techniques : Messages d'erreurs et Table des Codes ASCII, ainsi qu'un index alphabétique, complètent cet ouvrage.

De façon à éviter le maximum d'erreurs de transcription, les listings de programmes ont été directement photographiés. Pour permettre une meilleure lisibilité de ceux-ci, les libellés entre guillemets ont été écrits en minuscule, ainsi seules les instructions, noms de variables et de chaînes ... sont en lettres majuscules. Mais, compte tenu, de l'imprimante utilisée les accents et cédilles ont disparu ; nous prions le lecteur de bien vouloir nous excuser si quelques corrections manuelles ont été omises.

Ière Partie

Le BASIC Microsoft

Ecriture des programmes

Les numéros de ligne autorisés vont de 1 à 65529. Les lignes de programme peuvent être entrées dans un ordre quelconque.

Chaque ligne de programme ne doit pas excéder 255 caractères.

Le caractère "." permet dans une commande (LIST e.g.) de référencer la ligne courante.

'?' est équivalent à PRINT

Le caractère LINE-FEED permet d'écrire une même ligne de programme sur plusieurs lignes physiques.

Tout ce qui suit REM ou ' sur une ligne de programme est considéré comme commentaire.

Plusieurs instructions sur la même ligne sont séparées par ':'

Dès qu'une erreur de syntaxe est découverte dans une ligne pendant l'exécution d'un programme, le système BASIC affiche le N° de cette ligne et passe automatiquement en mode 'EDIT':

```
                10 BONJOUR
RUN
                SYNTAX ERROR IN 10
        10
```

L'utilisateur peut alors modifier la ligne. Mais s'il veut d'abord visualiser les valeurs des variables (une modification de ligne les remet à zéro), il frappe contrôle Q.

Un retour chariot permet également de quitter le mode 'EDIT'. Mais dans ce cas les variables sont remises à zéro.

Caractères de contrôle

contrôle-A : Positionne en mode EDIT sur la ligne courante.
contrôle-C : Interrompt l'exécution du programme en cours.
contrôle-G : Sonnerie.
contrôle-H : Supprime le caractère frappe.
contrôle-I : Tabulation (de 8 en 8).
contrôle-O : Suspend l'édition du programme pendant que le programme continue de s'exécuter. Un second contrôle-O réactive l'édition.
contrôle-R : Imprime la ligne courante.
contrôle-S : Suspend l'exécution du programme.
contrôle-Q : Réactive l'exécution suspendue.
contrôle-U : Annule la ligne courante.

EDITEUR DE PROGRAMME (MICROSOFT 5.)

Une ligne de programme peut être entièrement REECRITE.

Lors de l'introduction d'une ligne de programme, un caractère invalide peut être effacé par :

Contrôle H ou RUBOUT

Une ligne en cours d'introduction est entièrement effacée par :

Contrôle U

La commande 'EDIT N° ligne' permet de modifier une ligne déjà introduite

Ex : EDIT 100

Déplacement du curseur :	ESPACE	La frappe d'espaces déplace le curseur à DROITE 'xESPACE' déplace le curseur de x positions à droite
	RUBOUT	Déplace le curseur à GAUCHE 'xRUBOUT' déplace de x positions à gauche
Insertion de texte :	I (Insert)	Frapper I puis le texte à INSERER à la position courante Pour finir l'insertion faire ESCAPE puis continuer à déplacer le curseur ou faire RETOUR CHARIOT pour terminer l'édition de la ligne
Insertion en fin de ligne :	X	Positionne le curseur à la fin de la ligne puis met en mode insertion
Suppression de texte :	D (Delete)	La frappe de D supprime un caractère La frappe de xD provoque la suppression de x caractères
	H (Hack)	Supprime tous les caractères à DROITE de la position courante puis se positionne en mode insertion.
Recherche de texte :	S (Search)	'Scaractère cherché' positionne le curseur sur la première occurence du caractère cherché 'iScaractère cherché' positionne le curseur sur la ième occurence
	K (Search and Kill)	'iKcaractère cherché' positionné sur la ième occurence du caractère cherché mais supprime tous les caractères avant la position atteinte
Remplacement de texte :	C (Change)	'Ccaractère' change le caractère suivant en 'caractère' 'iCchaîne' change i caractères en 'chaîne' Après avoir frappé ces i caractères il y a retour en mode EDIT
Fin du mode EDIT	RC	La frappe de Retour chariot provoque l'impression du reste de la ligne La sauvegarde des modifications puis la fin du mode EDIT
	E (Exit)	Même résultat qu'avec RC mais il n'y a pas impression du reste de la ligne

Q (Cancel and Quit)	Provoque la sortie du mode EDIT mais sans sauvegarde des modifications
L (List)	Cette sous commande liste le reste de la ligne puis repositionne en début de ligne
A (Annule)	Annule les modifications et repositionne en début de ligne.

COMMANDES BASIC MICROSOFT 5

Ces commandes sont acceptées après affichage de OK (mode commande BASIC). Elles peuvent aussi être écrites comme instructions dans un programme.

AUTO *n° ligne départ, incrément* : génère un n° de ligne après chaque RC. Ce mode se termine par Contrôle C.

AUTO 100 positionne à 100 et incrémente de 10 en 10
AUTO 100,5 positionne à 100 et incrémente de 5 en 5

CLEAR *,adresse mémoire maxi,espace pour les chaînes* : efface toutes les variables en mémoire centrale.
Affecte l'adresse mémoire maxi pour le BASIC et réserve l'espace pour les chaînes qui est par défaut de 1000 caractères ou de 1/8 de l'espace mémoire disponible. (C'est la plus petit qui est considéré)

CLEAR efface les variables et réserve l'espace standard pour les chaînes
CLEAR ,,3000 efface les variables et réserve 3000 caractères pour les chaînes.

CONT : continue l'exécution d'un programme après arrêt de celui-ci par STOP,END ou par Contrôle C .N'est pas accepté si le programme a été modifié. (dans ce cas faire GOTO XX).

DELETE *n° ligne début-n° ligne fin* : supprime les instructions comprises entre les limites spécifiées. Les n° de lignes spécifiés doivent exister.

DELETE 100-200 supprime toutes les instructions comprises entre 100 et 200.

FILES "*U*:**x**.**x**" fournit la liste des fichiers de l'unité de disque spécifiée U=A,B,C,etc..

FILES "A:**x**.BAS" donne la liste de tous les fichiers du type BAS (BASIC) du disque A.

KILL "*U:Nom Fichier*" supprime un fichier de l'unité de disque spécifiée U=A,B,..

KILL "A:PAYE" supprime le fichier 'PAYE'

LIST *n° ligne départ-n° ligne fin* : liste les lignes du programme comprises entre les limites indiquées.
LIST liste tout le programme
LIST 10 liste la ligne 10
LIST 500- liste toutes les lignes à partir de 500
LIST -400 liste toutes les lignes jusqu'à 400.

LLIST Liste le programme en mémoire centrale sur le périphérique esclave.

LOAD "*U:nom fichier*",R charge en mémoire centrale un programme.R lance son exécution

```
LOAD "PAYE"     charge en mémoire centrale 'PAYE'
                de la disquette A
LOAD "B:PAYE"                  B
LOAD "PAYE",R                  A et exécute
```

NAME "*ancien nom-fichier*" AS "*nouveau nom-fichier*" : affecte un nouveau nom à un fichier.
NAME "PAYE" AS "PAIE"

NEW Annule le programme en mémoire centrale.

MERGE "*U:nom de fichier*" : concatène le programme spécifié (sauvegarde en ASCII). Les lignes du programme concaténé dont les N° sont identiques aux n° de ligne déjà en mémoire centrale remplacent les lignes en mémoire centrale.

RENUM *nouveau n° de ligne,première ligne,incrément* :
- 'nouveau n° de ligne' qui est de 10 par défaut spécifie le nouveau premier n° de ligne.
- 'première ligne' spécifie la première ligne où doit commencer la renumérotation. C'est la première ligne par défaut.
- 'incrément' représente l'incrément à utiliser pour la renumérotation. Il est de 10 par défaut.

```
RENUM            renumérote tout le programme avec
                 un nouveau n° de première ligne égal
                 à 10 et un incrément de 10.
RENUM 1000,,10   renumérote avec un nouveau premier
                 numéro de ligne égal à 1000 et un
                 incrément de 10.
RENUM 1000,900,10 renumérote à partir de la ligne 900
                 en lui affectant 1000 comme nouveau
                 numéro.
```

RUN *n° ligne* : exécute le programme en mémoire centrale au n° de ligne indiqué. Toutes les variables sont initialisées à zéro.

RUN exécute le programme à la première ligne.

RUN "*U:nom fichier*",R Clot tous les fichiers, supprime le contenu de la mémoire, charge le programme spécifié puis l'exécute. Si 'R' est spécifié, les fichiers ne sont pas clos.

SAVE "*U:nom fichier*" sauvegarde le programme de la mémoire centrale sur disque sous le nom indiqué .U=A,B,C,etc... représente l'unité de disquette.

```
SAVE "PAYE"      sauvegarde le programme sur l'unité A
SAVE "B:PAYE"    sauvegarde le programme sur l'unité B
```

SAVE "*U:nom fichier*",P sauvegarde le programme en mode 'protégé'. Le programme sauvegarde ne peut être listé.

SAVE "*U:nom fichier*",A sauvegarde le programme sous forme "ASCII", c'est-à-dire sous forme non compactée, permettant ainsi sa manipulation par l'éditeur de texte. Ce programme non compacté peut aussi être concaténé à un autre programme par la commande MERGE.

SYSTEM Permet de quitter le BASIC.

TRON Permet de visualiser à l'exécution du programme les N° des instructions effectivement exécutées. (TRACE)

TROFF Annule TRON.

VARIABLES

Noms

Les noms de variables s'écrivent avec plusieurs lettres ou chiffres mais le premier caractère doit être une lettre.

Avec les anciennes versions des BASICS MICROSOFT, seuls les 2 premiers caractères sont pris en considération (2 variables ayant des noms commençant par 2 caractères identiques sont dans ce cas connues comme une seule). Pour la version 5., les 40 premiers caractères sont significatifs.

Sur TRS 80 un nom de variable ne doit pas comporter de mot-clé du BASIC. Ainsi 'COIFFEUR' qui contient 'IF' n'est pas accepté.

Types

Il y a 4 types de variables : les variables *entières, simple précision, double décision et chaînes de caractères.*

Le type est précisé par un caractère à droite de la variable (% $ # !) :

Ex : X%=5 (variable du type entier)

Par défaut, lorsque le type n'est pas précisé, les variables sont en simple précision.

Type	Caractère de déclaration	Exemple
Entières (nombres entiers compris entre -32768 et +32767	%	X%=123:JOURS%=365
Simple précision (6 chiffres significatifs)	!	SOMME!=1234.56
Double précision (16 chiffres significatifs)	#	SOMME#=123456789.123
Chaînes de caractères de longueur variable (255 caractères au maximum)	$	NOM$="DUPONT"

Les variables de même nom avec des types différents sont considérées comme des variables distinctes. (AS# et A$ par ex).

CLEAR et RUN initialisent les variables avec des valeurs nulles (0 pour les variables numériques, chaîne vide pour les variables chaînes).

La place occupée en mémoire va, bien sûr, en croissant avec la précision des nombres. Il en va de même pour les temps de calcul sur ces nombres.

Constantes octales-hexadécimales

Des constantes octales sont spécifiées par '&' ou '&0' devant le nombre.

Ex : PRINT &0377 --→ 255

Des constantes HEXADECIMALES sont spécifiées par '&H' devant le nombre.

Ex : PRINT &HFF --➔ 255

Les nombres représentés vont de -32768 à +32767. Les nombres négatifs sont représentés sous forme de complément vrai. (le premier des 16 bits est positionné à 1 pour les nombres négatifs).

OCTAL	DECIMAL	HEXADECIMAL
&O100000	-32768	&H8000
&O100001	-32767	&H8001
&O177777	-1	&HFFFF
&O000000	0	&H0000
&O000001	+1	&H0001
&O777777	+32767	&H7FFF

Des constantes OCTALES/HEXADECIMALES définies dans des DATA pourront être lues ainsi :

```
10 DATA 377,&O377

20 READ X:PRINT VAL("&"+STR$(X)) ------> 255

30 READ X:PRINT X                ------> 255
```

DEFINT -DEFSNG - DEFDBL -DEFSTR
(définition globale de type de variables)

DEF*type lettres* permet de définir GLOBALEMENT le type de toutes variables dont les noms commencent par les lettres spécifiées, plutôt que de déclarer explicitement le type de chaque variable par un caractère (%,$,#,!).

On peut cependant déclarer explicitement par un caractère (!,%,#,$) un type de variable qui aurait été défini par DEFtype. La déclaration explicite est prioritaire.

DEFINT J	Toutes les variables commençant par J et non déclarées explicitement par un caractère (!,%,$,#) sont des variables entières.
DEFSTR A-C	Toutes les variables commençant par une lettre A,B ou C sont du type chaîne.
DEFINT I-N,R-T	Spécifie 2 domaines de variables entières.

La version interprétée du BASIC admet la redéfinition du type des variables en cours de programme. (la définition des variables ne peut être dynamique avec la version compilée de MICROSOFT 5.)

Conversion de types de variables

Le stockage d'une valeur se fait suivant le type de la variable.

```
10 A%=123.45
20 PRINT A%
RUN
 123
```

```
A#=1.001#    ---> A#=1.001
A#=1.001D    ---> A#=1.001
A#=1.001     ---> A#=1.00100046730042      Attention!
1.1 /2#      --->      .550000011920929
1.1#/2#      --->      .55
```

```
10 INPUT "Nombre? ";X#:PRINT X#
RUN
 Nombre? 1.001
 1.001
```

Lorsqu'une opération +,-,× doit être exécutée, les opérandes sont d'abord convertis avec la précision du plus précis des opérandes, puis l'opération est effectuée avec cette précision.

Ainsi le résultat d'une opération entre un opérande simple précision et un opérande double précision est en double précision.

Toutefois, si le résultat d'une opération sur 2 opérandes entiers (-32768 et 32767) n'est pas entier, c'est en simple précision qu'il est fourni.

Pour la division, les règles sont les mêmes sauf pour la division entre entiers où le résultat est fourni en simple précision.

```
10 A=2:B=3: R=A/B
20 A=2:B#=3:R#=A/B#
30 PRINT R,R#
RUN
 .6666667  .6666666666666667
```

Pour une opération de comparaison, les opérandes sont convertis à la même précision avant d'être comparés.

Effets de la conversion sur la précision :

La conversion en entier supprime la partie fractionnaire. Un nombre double précision converti en simple précision est arrondi.

Attention ! La conversion en double précision ne complète pas seulement par des zéros à droite.

```
10 A!=1.00001
20 B#=A!
30 PRINT A!,B#
RUN
 1.00001  1.000009894371033
```

On peut éviter cela en utilisant la fonction STR$

```
10 A!=1.00001
20 B#=VAL(STR$(A!))
30 PRINT A,B#
donne  1.00001  1.00001
```

EXPRESSIONS ET OPERATEURS

Une expression peut être simplement une constante du type numérique où chaîne, une variable ou une combinaison de constantes et de variables liées par des opérateurs.

Ex : (X*2)+4/6

Les opérateurs effectuent des opérateurs sur des valeurs. Ils sont classés en 3 catégories :

1/ Arithmétiques
2/ Relationnels
3/ Logiques

Opérateurs arithmétiques

^	Exponentiation
* /	Multiplication Division
+ -	Addition Soustraction
MOD \	Modulo Division entière

L'évaluation des expressions se fait avec l'ordre des priorités des opérateurs défini ci-dessus.

5+10/5 est égal à 7

Des parenthèses permettent de changer cet ordre. Ce sont d'abord les expressions entre parenthèses qui sont évaluées.

(5+10)/5 est égal à 3

X MOD Y : L'opérateur MOD donne le reste de la division de X par Y

PRINT 100 MOD 3 -------> 1

X\Y : L'opérateur \ donne la partie entière du quotient X par Y

PRINT 100 \ 3 -------> 33

(X doit être un entier compris entre -32768 et 32767).

Opérateurs relationnels

Ils comparent 2 valeurs. Le résultat est soit égal à -1 (condition vraie), soit égal à 0 (condition fausse).

Ceci permet de prendre une décision (par IF .. THEN ... ELSE)

```
100 IF X<0 THEN PRINT "NOMBRE NEGATIF" ELSE PRINT "NOMBRE POSITIF"
```

=	Egalité
<	Inférieur
>	Supérieur
<>	Différent
<=	Inférieur ou Egal

Opérateurs logiques

Ils testent des relations multiples. Le résultat de l'opération logique est soit (égal à 0) soit vrai (égal à -1) et permet ainsi de prendre une décision par IF .. THEN .. ELSE

```
100 IF A<0 AND B<0 THEN PRINT "A et B SONT NEGATIFS"
```

L'ordre des priorités d'évaluation des opérateurs logiques est le suivant : NOT - AND - OR - XOR - IMP - EQV .

Dans les tables de vérité ci-dessous '0' signifie 'FAUX' et '1' signifie 'VRAI'. Mais en réalité le TEST se fait sur 0 (FAUX) et sur >< 0(vrai) et le résultat est soit 0 (faux), soit -1 (vrai)

R1	R2	NOT (NON) NOT R1
1	-	0
0		1

AND (ET)		R1 AND R2	IMP (IMPLICATION)		R1 IMP R2
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	0	1	1
0	0	0	0	0	1

OR (OU)		R1 OR R2	EQV (EQUIVALENT)		R1 EQV R2
1	1	1	1	1	1
1	0	1	1	0	0
0	1	1	0	1	0
0	0	0	0	0	1

XOR (OU EXCLUSIF)		R1 XOR R2
1	1	0
1	0	1
0	1	1
0	0	0

Les priorités d'évaluation des expressions comportant des opérateurs arithmétiques, relationnels et logiques sont :

1/ Parenthèses
2/ Opérateurs arithmétiques
3/ Opérateurs relationnels (1 seul niveau)
4/ Opérateurs logiques

```
Ex: MXMUN=-((A>B)*A+(A<=B)*B)        donne le plus GRAND de A et B
```

Opérateurs booléens

La manipulation de bits et les opérations booléennes sur ces bits s'effectuent avec les opérateurs AND, OR, NOT, ...

Ces derniers opèrent sur des groupes de 16 bits au plus qui sont spécifiés par des nombres allant de -32768 à 32767 (représentés en complément vrai de façon interne). Les opérations s'effectuent BIT à BIT.

```
Ex:    15  -------->0000000000001111
        4  -------->0000000000000100
  15 AND 4 ------->0000000000000100   ------>4

Ex:     4  -------->0000000000000100
        2  -------->0000000000000010
   4  OR 2 ------->0000000000000110   ------->6

Ex:    -1  -------->1111111111111111
        8  -------->0000000000001000
  -1 AND 8 ------->0000000000001000   ------>8
```

ENTREES AU CLAVIER : INPUT - LINE INPUT - INPUT$

INPUT "MESSAGE" ; **variable 1, variable 2, ...**

Permet d'entrer, pendant l'exécution d'un programme, une ou plusieurs valeurs numériques ou chaînes de caractères. Les variables spécifiées dans l'ordre INPUT sont séparées par des virgules.

Après l'affichage du message, l'opérateur doit entrer les valeurs des variables dans l'ordre défini par l'instruction INPUT en les séparant par des virgules dans l'ordre défini par l'instruction INPUT en les séparant par des virgules.

Les variables numériques peuvent également être séparées par des espaces.

```
Ex:   10 INPUT "Nom,Prenom,Age ? ";NOM$,PRENOM$,AGE
```

Après l'affichage du message 'Nom ,Prénom ,Age ?' l'opérateur entre :

```
DUPONT,JEAN,20
```

Les variables NOM$,PRENOM$,AGE prennent alors respectivement les valeurs DUPONT, JEAN et 20

Lorsque l'opérateur veut affecter une valeur nulle à une variable, il frappe seulement un retour-chariot ou une virgule. (sur TRS 80, une variable conserve son ancienne valeur si on entre seulement un retour-chariot).

Si l'opérateur n'entre pas toutes les valeurs prévues, le système envoie des ?? pour signifier à l'opérateur qu'il doit entrer les valeurs manquantes.

Si l'opérateur entre une chaîne de caractères alors que c'est une valeur numérique qui est attendue, le message ?REDO est envoyé. (Recommencez depuis le début). Les chaînes de caractères qui comprennent des virgules doivent être entrées entre guillemets.

```
        10 INPUT "Rue? ";RUE$
        20 PRINT RUE$
RUN
           Rue? "9,rue d'Orgemont"

           9,rue d'Orgemont
```

LINE INPUT "MESSAGE"; **variable chaîne**

Permet de lire toute une chaîne au clavier, sans tenir compte des séparateurs tels que la virgule comme c'est le cas avec l'instruction INPUT. C'est seulement un retour-chariot qui délimite la fin de la chaîne qui ne peut cependant pas excéder 255 caractères.

INPUT$(1):(MICROSOFT 5.)

Fournit au programme chaque caractère frappé au clavier, sans attendre de séparateur (comme pour INPUT) et permet ainsi de mieux contrôler l'introduction des données par l'opérateur. Le caractère acquis doit être imprimé par programme (s'il est valide).

On peut regretter que cette instruction soit 'bloquante', c'est à dire que le programme doit attendre qu'un caractère soit frappé.

```
100 Y$=""                          ' Y$:chaîne acquise
110 X$=INPUT$(1)                   ' Lecture de 1 caractère
120 IF ASC(X$)=13 THEN GOTO 150    ' Retour chariot?
125 IF ASC(X$)=8 THEN
        Y$=LEFT$(Y$,LEN(Y$)-1):PRINT X$;:GOTO 110   ' Curseur gauche?
130 PRINT X$;                      ' Impression du caractère acquis
140 Y$=Y$+X$:GOTO 110              ' Ajout du caractère acquis
150 ' Traitement de la chaîne Y$
```

Sur TRS 80 INKEY$ (l'équivalent de INPUT$(1)), teste le clavier puis retourne immédiatement soit le caractère frappé soit une chaîne nulle si aucun caractère n'a été frappé.

Ceci permet d'effectuer éventuellement un traitement (par appel d'un sous-programme) en attendant la frappe d'un caractère :

```
100 X$=INKEY$                          ' Lecture du clavier
110 IF X$="" THEN GOSUB ....:GOTO 100  ' Appel d'un sous-programme
115 '                                  ' en attendant la frappe
116 '                                  ' d'un caractère au clavier.
120  Traitement du caractère acquis
125 '
130 GOTO 100
```

IF .. THEN .. ELSE .. (SI .. ALORS .. SINON ..)

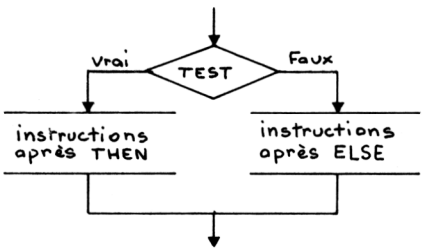
IF *expression logique* THEN *suite d'instructions* ELSE *suite d'instructions*

vraie ou fausse si exp. logique vraie si exp. logique fausse

Cette instruction teste si une expression logique est vraie ou fausse.

Si celle-ci est vraie, alors toutes les instructions comprises entre THEN et ELSE, sont exécutées.

Sinon, ce sont toutes les instructions après le ELSE qui sont exécutées.




```
Ex:     10 INPUT "Nombre A,Nombre B ? ";A,B
        20 IF A>B THEN PRINT "A > B"  ELSE PRINT "A <= B"
        30 GOTO 10
```

En fait, le test peut se faire, non seulement sur une expression logique, mais aussi sur une expression arithmétique

qui est interprétée comme fausse si elle a une valeur nulle ou comme vraie pour toute autre valeur. Mais on évitera d'utiliser cette particularité du BASIC.

```
Ex: 10 IF I THEN PRINT "I est different de 0"

Ex: 10 IF (I<10)*(I>20) THEN PRINT " I n'est pas compris entre 10 et 20"
```

Les IF .. THEN .. ELSE .. peuvent être imbriqués, mais il faut alors bien s'assurer qu'à chaque IF-THEN il correspond un ELSE.

```
10 INPUT "Nombre A,Nombre B ? ";A,B
20 IF A<=B THEN IF A<B THEN PRINT "A<B" ELSE PRINT "A=B" ELSE PRINT "A>B"
```

FOR .. NEXT ...STEP ..(BOUCLES)

FOR *variable compteur=valeur initiale* TO *valeur limite*
STEP *incrément*

:
:

NEXT *variable compteur*

La valeur initiale, la valeur limite et celle de l'incrément peuvent être spécifiées par des expressions.

Les instructions qui suivent FOR sont exécutées d'abord avec pour valeur de variable la valeur initiale spécifiée dans FOR.

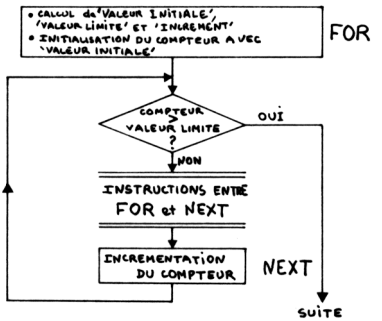
Lorsque l'instruction NEXT est rencontrée, la valeur de la variable compteur est incrémentée de la valeur spécifiée dans STEP (1 par défaut), puis est comparée à la valeur limite indiquée dans FOR :

- Si la nouvelle valeur de la variable compteur reste inférieure ou égale (pour un incrément positif) à la valeur limite spécifiée après TO, il y a branchement à l'instruction après FOR et les instructions jusqu'à NEXT sont à nouveau exécutées avec la nouvelle valeur de la variable compteur.

- Si au contraire, la valeur de la variable compteur dépasse (pour un incrément positif) la valeur limite, le programme continue à s'exécuter après l'instruction NEXT.

Lorsque l'incrément spécifié dans STEP est négatif, c'est au moment où la valeur de la variable compteur devient inférieure à la valeur limite que l'exécution de la boucle FOR ..NEXT se termine.

```
Ex:  10 FOR I=10 TO 1 STEP-1
     20   PRINT I;
     30 NEXT I

     40 PRINT "FIN"
run
     9 8 7 6 5 4 3 2 1 FIN
```



```
10 FOR I=0 TO 1 STEP .3
20   PRINT I;
30 NEXT I
run
 0 .3 .6 .9
```

Calculées au moment de l'exécution de l'instruction FOR, donc une seule fois, *'valeur initiale'*, *'valeur limite'* et *'incrément'* ne varient pas en cours d'exécution de la boucle si les valeurs des variables qui ont servi à les calculer évoluent pendant l'exécution. En revanche, la valeur de la variable compteur peut être modifiée.

Depuis la version 5., c'est en début de boucle que se fait la comparaison de la valeur de la variable compteur à la valeur finale.

La boucle n'est donc pas exécutée une fois si la valeur finale est inférieure à la valeur initiale (pour un incrément positif) comme c'était le cas lorsque le test se faisait en fin de boucle (versions antérieures à la 5.).

La variable compteur doit être du type simple précision ou entier. Avec ce dernier type, l'exécution est plus rapide.

Boucles imbriquées

Plusieurs boucles FOR .. NEXT peuvent être 'emboîtées', c'est à dire qu'une boucle peut être placée à l'intérieur d'une autre. Mais il est interdit de les faire se chevaucher.

Exemple :

```
10 FOR I=1 TO 5          ' Boucle externe

15   PRINT "I=";I;"J=";

20   FOR J=1 TO 10       ' Boucle interne
30     PRINT J;
40   NEXT J

50   PRINT

60 NEXT I


   run
        I=1 J= 1 2 3 4 5 6 7 8 9 10
        I=2 J= 1 2 3 4 5 6 7 8 9 10
         ........................
        I=5 J= 1 2 3 4 5 6 7 8 9 10
```

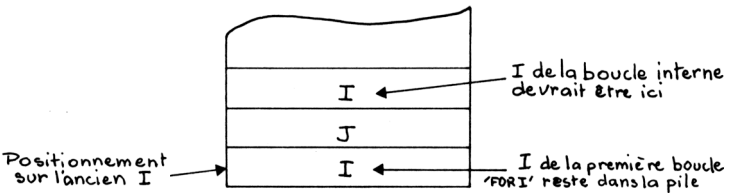
S'il n'y a pas d'instruction entre NEXT J et NEXT I, NEXT J,I donne le même résultat.

NEXT (au lieu de NEXT J) est accepté, puisqu'en fait NEXT incrémente le compteur du FOR le plus récent et que celui-ci est supprimé dès qu'il atteint la valeur limite. Mais pour des raisons de lisibilité, on indiquera le nom de la variable.

Particularité de la BOUCLE FOR pour les BASICS INTERPRETES :
(pour versions antérieures à la 5.)

Ce programme apparemment bien écrit ne fonctionne pas :

```
10 FOR I=1 TO 5
20   IF I=3 GOTO 40            ' On sort par GOTO
30 NEXT I

40 FOR J=1 TO 10               ' Boucle externe

50   FOR I=1 TO 6              ' Boucle interne
60     PRINT "I=";I
70   NEXT I

75   PRINT "Attention ! il va y avoir le message
                NEXT WHITOUT FOR"
80 NEXT J
```



Ceci s'explique de la façon suivante :

- A chaque fois qu'un FOR est rencontré, la variable compteur correspondante est stockée dans une pile. Dès que la variable compteur atteint sa limite par NEXT, elle est supprimée de la pile (ainsi que celles du dessus).

- Mais lorsqu'on quitte une boucle FOR par 'GOTO', la variable compteur reste dans la pile et si une nouvelle instruction FOR pour cette même variable est rencontrée, le pointeur de pile se positionne sur l'ancienne variable. Par conséquent, si cette variable compteur correspond à une BOUCLE INTERNE à une autre boucle, la variable de la boucle externe (au-dessus dans la pile) est éliminée.

- Par contre, si la variable compteur restée dans la pile est utilisée à nouveau dans une boucle externe s'achevant normalement, il n'y a pas de problème.

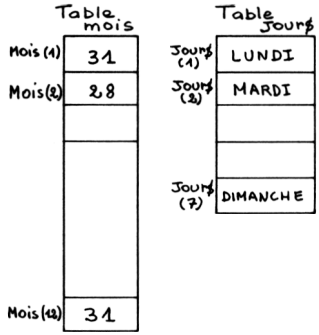
Solution : On peut quitter la boucle par :
IF .....THEN I=*valeur limite*:NEXT I:GOTO XX

On peut également prévoir, pour des boucles ne s'achevant pas complètement, des variables particulières qui ne seront pas utilisées dans des boucles internes.

La séquence ci-dessous ne pose pas de problème ; la fin de la boucle 'FOR I' élimine de la pile le compteur de la boucle interne 'FOR J'

```
10 FOR I=1 TO 5
20   FOR J=1 TO 4
30     IF J=2 THEN GOTO 45
40   NEXT J
45   PRINT "J=";J
50 NEXT I
```

TABLEAUX : DIM - ERASE



Un tableau est une suite ordonnée d'éléments. Pour les tableaux à une dimension, celle-ci est déclarée par une instruction.

DIM NOM-TABLEAU(*Nombre éléments*)

Ainsi une table contenant les nombres de jours des différents mois de l'année peut être déclarée par :

```
DIM MOIS(12)
```

Tous les types de tableaux sont acceptés (numériques, chaînes de caractères). Les règles d'appellation sont les mêmes que pour les variables.

Exemple :

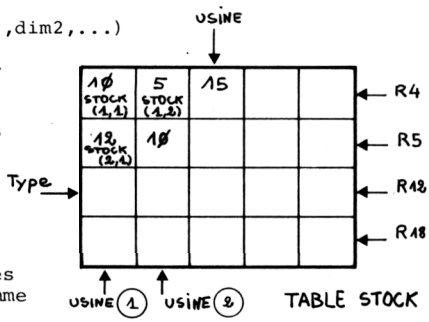
```
X=7
DIM JOUR$(X)
```

(la version compilée n'accepte qu'une constante dans une déclaration de dimension).

La déclaration des tables de dimension inférieure à 10 n'est pas nécessaire. Elle est toutefois recommandée, à la fois par souci de clarté et d'économie de la mémoire. Bien que rarement utilisé, l'élément 0 d'une table existe.

Plus généralement, les tables peuvent avoir plusieurs dimensions déclarées par :

```
DIM NOM-TABLE(dim1,dim2,...)
```



Soit une table à 2 dimensions contenant des stocks de voitures de différents types (4) et de différentes usines (5) pour chacun de ces types.

DIM STOCK(4,5) réserve une table de 4 lignes et 5 colonnes.

L'introduction des données dans cette table se programme ainsi :

```
10 DIM STOCK(4,5)

20    FOR TYPE=1 TO 4

30      FOR USINE=1 TO 5
40        PRINT "Stock:";"Type ";TYPE;"Usine ";USINE;
50        INPUT STOCK(TYPE,USINE)
60      NEXT USINE

70    NEXT TYPE

RUN
        Stock:Type 1 Usine 1 ?        ' L'opérateur entre la valeur
        Stock:Type 1 Usine 2 ?        ' L'opérateur entre la valeur
            .
            .
```

Pour connaître le nombre de voitures d'un type nous écrirons :

```
100 INPUT "Type? ";TYPE

110 TOTAL=0

120 FOR USINE=1 TO 5
130   TOTAL=TOTAL+STOCK(TYPE,USINE)
140 NEXT USINE

150 PRINT "Total TYPE:";TYPE;"=";TOTAL
160 GOTO 100
```

Le total des voitures d'une usine sera donné par :

```
200 INPUT "Usine? ";USINE

210 TOTAL=0

220  FOR TYPE=1 TO 4
230    TOTAL=TOTAL+STOCK(TYPE,USINE)
240  NEXT TYPE

250 PRINT "Total USINE:";USINE;"=";TOTAL
260 GOTO 200
```

Comme pour les tables à 1 dimension, la déclaration d'une table à plusieurs dimensions, toutes inférieures ou égales à 10, n'est pas nécessaire.

Alors qu'une table et une variable peuvent avoir un même nom, plusieurs tables, même si elles ont des nombres de dimensions différents, ne peuvent avoir le même nom :

Exemple :

```
10 DIM A(7):DIM A(7,8)   est interdit
```

ERASE *Nom de tableau,.....* : *supprime un ou plusieurs tableaux de la mémoire centrale*

```
100 ERASE A,NOM$  'Efface le tableau A et le tableau NOM$
110 DIM A(200)
```

La place mémoire est récupérée dynamiquement. Un effacement de tableau non existant provoque une erreur.

(ERASE n'est pas accepté avec la version compilée de MBASIC 5.)

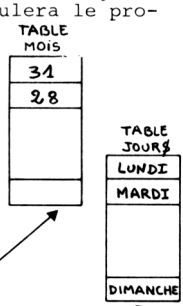
```
10 '
20 '                SAISIE  D'UN TABLEAU 2 DIMENSIONS
30 '                ================================
40 '
50 '              R :              Retour à la  zone précédente
60 '              RT:              Retour au bloc précédent
70 '              Retour Chariot: N'affecte pas la valeur de la zone
80 '
90 DATA "FLINS   ","CLEON   ",BILLANCOURT,"SANDOUVILLE","DOUAI  ","ORLEANS":
        FOR I=1 TO 6:READ USINE$(I):NEXT I
100 DATA R5,R6,R12,R18,R20,R30,R40:FOR I=1 TO 7:READ TYPE$(I):NEXT I
110 '
120 DIM STOCK(7,6)                            ' 7 Types - 6 Usines
125 '-----------------------------------------------------------------------
130 FOR TYPE=1 TO 7                           ' Saisie tableau
140 '
150    FOR USINE=1 TO 6
160       PRINT "Stock:";TYPE$(TYPE);"-";USINE$(USINE);       ' Libellé
170       PRINT TAB(21) STOCK(TYPE,USINE) TAB( 25)            ' Ancienne valeur
180       INPUT X$:X=VAL(X$):IF X<0 OR X>100 THEN GOTO 160    ' Nouvelle valeur
190       IF X$="R" AND USINE>1 THEN USINE=USINE-1:PRINT:GOTO 160
200       IF X$="RT" AND TYPE>1 THEN TYPE=TYPE-2:PRINT:GOTO 250
210       IF X$><"" THEN STOCK(TYPE,USINE)=X                  ' Saut de la zone
220    NEXT USINE
230 '
240    PRINT
250 NEXT TYPE
260 '-----------------------------------------------------------------------
270 '
280 ' run
290 '
300 '   Stock:R5-FLINS        0  ? 5
310 '   Stock:R5-CLEON        0  ? 7
320 '        .
330 '        .
```

DATA-READ-RESTORE

***DATA* valeur 1, valeur 2,...**

Les instructions 'DATA' permettent de définir de façon 'statique' des données (constantes) que manipulera le programme (ces 'DATA' ne peuvent qu'être lues)

```
10 DATA 31,28,30,31 .....30,31
20 DATA LUNDI,MARDI,......,DIMANCHE
```



***READ* variable**

Affecte séquentiellement aux variables les valeurs définies dans DATA.

```
100 FOR I=1 TO 12
110   READ MOIS(I)
120 NEXT I                  ' Lecture dans table 'MOIS'

200 FOR I=1 TO 7
210   READ JOUR$(I)
220 NEXT I                  ' Lecture dans table 'JOUR$'
```

La première instruction 'READ *variable*' exécutée affecte à cette variable la valeur de la seconde donnée définie dans DATA.

La seconde instruction 'READ *variable*' exécutée affecte à cette variable la valeur de la seconde donnée définie dans DATA etc.

Toutes les DONNEES définies dans les DATA sont ainsi lues séquentiellement au fur et à mesure de l'exécution d'instructions READ, dans l'ordre des numéros de ligne.

Les types des données doivent bien sûr s'accorder avec les types des variables pour lesquelles elles sont lues.

Il convient d'être soigneux dans l'écriture des DATA ; en effet un simple décalage d'une donnée fausserait tout.

Aussi, afin de clarifier les programmes, les DATA seront de préférence lus dès le début du programme dans des tables auxquelles il sera ensuite possible d'accéder de façon directe.

Au moment de la mise au point, on visualisera la valeur de la variable qui contient la dernière DATA lue afin de s'assurer du bon cadrage.

Les DATA, quelles que soient leur implantation physique dans le programme, sont vues comme un ensemble unique. (Sauf si RESTORE X est utilisé).

Les données numériques qui doivent être lues comme des chaînes sont écrites entre des guillemets.

```
DATA "3","6","8":FOR I=1 TO 3:READ X$(I):NEXT I
```

Les données chaînes de caractères comportant des caractères spéciaux tels que ': ,' doivent également être écrites entre guillemets.

```
20 DATA "LUNDI:","MARDI:",.....
```

RESTORE

Si les DATA doivent être lues à nouveau, il faut utiliser RESTORE.

RESTORE *n° ligne* provoque un positionnement sur les DATA de la ligne spécifiée :

```
 10 DATA 3,4,5,6
 20 RESTORE 100          ' Positionne sur la ligne 100
 50 READ X:READ Y:READ Z
 60 PRINT X,Y,Z
100 DATA 7,8,9
run
      7  8  9
```

LES CHAINES DE CARACTERES

.LEFT$ RIGHT$ MID$
.LEN
.STR$ VAL
.ASC CHR$
.INSTR
.STRING$ SPACE$
.OCT$ HEX$
.FRE(X$)

L'espace réservé pour les chaînes de caractères est par défaut de 1 000 caractères ou de 1/8e de l'espace mémoire disponible (c'est le plus petit qui est considéré), voir CLEAR.

L'affectation d'une valeur à une chaîne s'écrit :

nom de chaîne="suite de caractères"

Exemple :

```
10 NOM$="DUPONT"
```

Les "" indiquent que DUPONT doit être interprété comme une chaîne de caractères et non comme une variable. La longueur d'une chaîne de caractères, qui n'a pas à être déclarée, peut varier en cours d'exécution du programme (jusqu'à 255). De même, la longueur de chaque élément d'une table de chaînes peut varier dynamiquement.

La concaténation (réunion) de chaînes de caractères est réalisée par l'opérateur noté "+"

```
10 NOM$="DUPONT"
20 PRENOM$="JEAN"
30 NP$=NOM$+PRENOM$
40 PRINT NP$
RUN
   DUPONTJEAN
```

Comparaisons

La comparaison de chaînes de caractères se fait avec les opérateurs :

= < > <> >= <=

Les chaînes sont comparées caractère par caractère de la gauche vers la droite jusqu'à ce que l'un des caractères d'une chaîne soit plus grand que l'autre (code ASCII supérieur). C'est alors cette chaîne qui est considérée comme la plus grande ('DURAND' est plus grand que 'DUPONT').

Si tous les caractères sont égaux, les chaînes sont considérées comme égales.

Exemple :

```
10 INPUT "Reponse (OUI-NON) ? ";R$
20 IF R$="OUI" THEN PRINT "Vous avez dit OUI":GOTO 60
30 IF R$="NON" THEN PRINT "Vous avez dit NON":GOTO 60
40 IF R$=""    THEN PRINT "Vous n'avez entre qu'un RC":GOTO 60
50 PRINT "Je n'ai  pas compris"
60 GOTO 10
```

Remarque : Sur TRS 80, la ligne 10 doit être remplacée par :

R$="":INPUT .... pour que la ligne 40 fonctionne comme prévu (voir INPUT)

LEFT$, RIGHT$, MID$

LEFT$,RIGHT$,MID$ permettent d'accéder respectivement aux caractères de gauche, de droite et de l'intérieur d'une chaîne.

LEFT$ (CHAINE, *longueur à prendre à gauche)*
RIGHT$ (CHAINE, *longueur à prendre à droite)*
MID$ (CHAINE, *position début, longueur à prendre)*

'CHAINE', *'longueur à prendre'* et *'position début'* peuvent être des expressions.

Les valeurs de *'longueur à prendre'* et de *'position début'* doivent être comprises entre 0 et 255.

Ex: SX$=LEFT$(X$,2)

```
10 X$="MRBALUJEAN"
20 SX$  =LEFT$ (X$,2)          ' 2 caractères à gauche
30 PREN$=RIGHT$(X$,4)          ' 4 caractères à droite
40 NOM$ =MID$  (X$,3,4)        ' 4 caractères à partir du 3ème
50 PRINT SX$,PREN$,NOM$
RUN
   MR    JEAN  BALU
```

Ex:

```
10 INPUT "Reponse? (OUI-NON)  ";R$
20 IF LEFT$(R$,1)="O" THEN GOTO .... Teste si l'opérateur
                                     a répondu 'OUI'
```

Ex:

```
10 NOM$="DUBONET"

20 FOR I=1 TO 7
30   PRINT LEFT$(NOM$,I)
40 NEXT I
RUN
   D
   DU
   DUB
   DUBO
   DUBON
   DUBONE
   DUBONET
```

Si *'longueur à prendre'* spécifiée est plus grande que la longueur de la chaîne, le résultat est la chaîne elle-même.

Lorsque *'longueur à prendre'* n'est pas précisée dans MID$, cette fonction devient équivalente à RIGHT$.

Si *'position début'* spécifiée dans MID$ est plus grande que la chaîne elle-même, une chaîne vide est retournée.

MID$(CHAINE1*, position début, longueur à remplacer)*=CHAINE2

Permet, à partir de *'position début'* dans 'CHAINE1" et sur la longueur spécifiée, de remplacer des caractères par ceux de 'CHAINE2'

Exemple :

```
60 MID$("MRBALUJEAN",3,4)="XXXX" ----> X$="MRXXXXJEAN"
```

Attention : ne permet pas l'insertion ou la suppression de caractères mais seulement la substitution.

Si *'CHAINE2'* est plus longue que *'longueur à remplacer'*, seuls les premiers caractères de *'CHAINE2'* sont pris en considération.

Si *'CHAINE2'* est plus courte que *'longueur à remplacer'*, il n'y a substitution des caractères que sur une longueur égale à celle de *'CHAINE2'*.

```
10 MIDØ("123456",3,2)="XXX" ---> XØ='12XX56'
10 MIDØ("123456",3,2)="X"   ---> XØ='12X456'
```

LEN (chaîne)

Donne la longueur d'une expression chaîne.

```
10 NOM$="DUPONT"
20 L=LEN(NOM$)
30 PRINT L
RUN
   5
```

STRØ(X)

Convertit une expression numérique X en une chaîne de caractères.

```
10 X=123
20 X$=STR$(X)
30 PRINT X$,LEN(X$)
RUN
   123     4
```

Remarque : Le premier caractère de la chaîne est réservé pour le signe : C'est un espace pour un nombre positif et un signe '-' pour un nombre négatif.

PRINT MIDØ(STRØ(X),2,1) donne 1 comme résultat

VAL (chaîne)

Fonction inverse de STRØ, elle donne la valeur numérique d'une expression chaîne.

```
10 X$="123 FRANCS""
20 X =VAL(X$)
30 PRINT X
RUN
   123
```

Si le premier caractère n'est pas un caractère décimal, un espace, un signe +, un signe - ou '.' , le résultat est égal à zéro.

```
VAL(" 123") ---> 123
VAL("+123") ---> 123
VAL("ABCD") ---> 0
VAL(" +123")---> 0
```

ASC (caractère)

Chaque caractère a un code interne (code ASCII) auquel on accède par la fonction ASC (caractère)

```
10 X$="A"
20 X=ASC(X$)
30 PRINT X
RUN
   65
```

***ASC* (chaîne)**

Donne le code ASCII du premier caractère d'une expression chaîne.

```
PRINT ASC("BONJOUR") ----> 66
```

Une chaîne nulle comme argument provoque une erreur.

CHRØ(X)

Fonction inverse de la fonction ASC, elle permet de générer des caractères ayant pour code ASCII la valeur de X. Cette valeur doit être comprise entre 0 et 255. X peut être une constante, une variable ou une expression.

```
10 FOR I=65 TO 65+26
20 PRINT CHR$(I);
30 NEXT I
RUN
        ABCDEFGHIJ.....Z
```

Conversion MAJUSCULES-minuscules

```
PRINT CHR$(ASC("A")+32) ----> a  (voir table des codes ASCII)

10 INPUT "Votre NOM? ";NOM$
20 FOR I=1 TO LEN(NOM$)
30 PRINT CHR$(ASC(MID$(NOM$,I,1))+32);
40 NEXT I
RUN
        Votre NOM? ROULET

        roulet
```

Caractères spéciaux (de contrôle)

```
PRINT CHR$(7)    ------> active la sonnerie du terminal
PRINT CHR$(13)   ------> provoque un retour en debut de ligne
                         (sans saut de ligne)
PRINT CHR$(10)   ------> provoque  un saut de ligne
```

La fonction CHRØ(X) est notamment utilisée pour l'envoi de caractères spéciaux aux terminaux (écran, imprimante, tables traçantes, ..). Elle permet de faire apparaître explicitement ces caractères dans la liste du programme.

La frappe directe au clavier, de caractères de contrôle, si elle est possible, aurait 2 effets :

- au moment où le programme est listé, les caractères de contrôle seraient envoyés au terminal qui les interprêterait (par ex. la sonnerie serait activée par 'contrôle G' (code ASCII 7)).

- Les caractères de contrôle n'apparaîtraient pas sur la liste du programme.

Exemples divers

Suppression d'un caractère à droite d'une chaîne :

```
10 X$="DUPONT"
20 X$=LEFT$(X$,LEN(X$)-1)
30 PRINT X$

RUN
        DUPON
```

Normalisation d'une chaîne à une longueur fixe :

```
10 X$="DUPONT"
20 Y$=RIGHT$("          "+X$,10)
30 PRINT Y$,LEN(Y$)
RUN
          DUPONT     10
```

Insertion d'un caractère dans une chaîne :

```
10 Z$="AAAAAAA"
20 X$="B":L=3
30 Z$=LEFT$(Z$,L)+X$+RIGHT$(Z$,LEN(Z$)-L)
40 PRINT Z$
RUN
   AAABAAAA
```

Remplissage par des zéros à gauche :

```
10 X=123
20 X$=RIGHT$(STR$(100000+X),5)
30 PRINT X$
RUN
   00123
```

***INSTR* (position départ, chaîne, chaîne cherchée)**

Recherche la position d'une chaîne dans une autre. Par défaut, la position de départ est égale à 1 :

- Si la chaîne cherchée n'est pas trouvée, le résultat est égal à 0.
- Si la chaîne cherchée est nulle, le résultat est la position de départ spécifiée.
- Si la position de départ est supérieure à la longueur de la chaîne où s'effectue la recherche, le résultat est nul.

Ex:

```
10 X$="DUPONT.JEAN"
20 X=INSTR(X$,".")      ' Recherche du caractère '.'
30 Y$=LEFT$(X$,X-1)
40 PRINT Y$,X
RUN
   DUPONT      7
```

Ex: `PRINT INSTR(3,"ABAABAAA","B")` -------> 5

Exemples d'applications de INSTR

Conversion hexadécimal-décimal

```
10 X$="C"
20 X=INSTR("0123456789ABCDEF",X$)-1
30 PRINT X        ------>12
```

Conversion alphabétique-numérique

Cette séquence d'instructions permet de répondre à une question 'Mode?' non pas par un chiffre, mais par une lettre (plus mnémonique).

```
10 INPUT "Mode? (C,A,P,R,N,M) ";MODE$      ' Entrer C,A,P,...
20 X=INSTR(" CAPRMN",MODE$)                ' Position du caractère frappé?
30 ON X GOTO 10,200,250,300,500,700,750    ' Branchement multidirections
40 GOTO 10                                 ' X=1 (chaîne vide)
```

Remarque : Si MODE$ est une chaîne vide, X est égal à 1. C'est ce qui explique la présence d'un espace dans l'instruction 20 devant 'CAPRMN'

Vérification de caractères

Vérifie si un caractère frappé appartient bien à un ensemble de caractères autorisés : (chiffres plus le '.' sur l'exemple).

```
 10 X$=INPUT$(1)                  ' Introduction de 1 caractère
 20 X=INSTR("0123456789.",X$)     ' Position du caractère frappé?
 30 IF X=0 THEN GOTO 10           ' Caractère invalide?
 40 Y$=Y$+X$                      ' Acquisition du caractère
 50 PRINT X$;                     ' Impression du caractère acquis

100 GOTO 10
```

STRING$ (nombre de fois, chaîne)

Génère une chaîne de caractères égale à la chaîne spécifiée, multipliée par le nombre de fois indiqué.

```
Ex:   10 X$=STRING$(10,".")        ' Genere une chaine de 10 '.'
      20 PRINT X$
 RUN
          ..........
```

Le programme suivant complète des libellés par des '.' à droite :

```
10 NZ$(1)="NOM:"
20 NZ$(2)="PRENOM:"
30 NZ$(3)="DATE NAISS:"

50 FOR I=1 TO 3
60 PRINT NZ$(I);STRING$(20-LEN(NZ$(I)),".");:INPUT X$
70 NEXT I

RUN
        NOM:..............
        PRENOM:...........
        DATE NAISS:.......
```

SPACE$(X)

Génère une chaîne de X espaces. X peut être une expression

```
10 X$="DUPONT"+SPACE$(10)+"JEAN"
20 PRINT X$
RUN
        DUPONT          JEAN
```

Notons que la fonction SPC ne génère pas une chaîne d'espaces et n'est utilisable qu'avec l'instruction PRINT.

OCT$ (expression)

Génère une chaîne qui représente la valeur octale de l'argument décimal.

```
PRINT OCT$(24)  ----> 30
```

HEX$ (expression)

Génère une chaîne qui représente la valeur hexadécimale de l'argument décimal.

```
PRINT HEX$(26)   ----> 1A
```

FRE(X$)

Fournit l'espace libre pour les chaînes (X$ est un argument fantôme).

```
PRINT FRE(X$)    -----> 2300
```

EDITIONS

PRINT	POS LPOS
TAB	LPRINT
SPC	WRITE
PRINT USING	WIDTH

L'édition n'est sans doute pas la partie la plus noble de l'informatique, mais il faut bien, à l'issue d'un traitement, éditer des résultats, et la qualité de cette "édition" doit être, pratiquement, toujours excellente.

PRINT expression

Un simple PRINT d'une constante, variable ou expression affiche ou imprime (selon qu'il s'agit d'un écran ou d'une imprimante) la valeur de celle-ci puis provoque un retour en début de ligne et un saut de ligne.

```
10 X=123
20 PRINT X
RUN
   123
```

PRINT,

L'impression de plusieurs valeurs sur une même ligne peut se faire simplement en séparant dans l'instruction les noms des variables ou les expressions par des virgules.

```
10 SOMME=200:NOMBRE=10
20 PRINT SOMME,NOMBRE,SOMME/NOMBRE,1980
RUN
   200        10        20        1980
```

Mais dans ce cas, l'impression des valeurs est faite selon un format standard (colonnes 1, 15, 29.. etc., les valeurs sont cadrées à gauche).

PRINT ;

Un point-virgule en fin d'instruction PRINT empêche le saut de ligne : les chaînes de caractères sont concaténées, les valeurs numériques sont suivies par un espace et précédées soit par un espace pour les nombres positifs, soit par le signe '-' pour les nombres négatifs.

```
10 NOM$="DUPONT":PRENOM$="JEAN"
30 PRINT NOM$;
40 PRINT PRENOM$
RUN
   DUPONTJEAN

PRINT "DUPONT";" ";"JEAN"     ------>  DUPONT JEAN
PRINT 123;456;- 789           ------>   123  456 -789
```

PRINT TAB(X)

La fonction TAB(X) permet de positionner directement la tête d'impression à l'intérieur d'une ligne en colonne X. X peut être une expression.

```
     10 A=123 : B=456
     20 PRINT A TAB(15) B
RUN
        123         456
                     ↑
                colonne 15
```

Attention : La tête d'impression ne peut revenir en arrière. Si X spécifié est plus petit que la position courante de la tête, TAB(X) est sans effet.

PRINT SPC(X)

Avec la fonction SPC(X), X blancs sont imprimés à partir de la position courante :

```
     10 NOM$="DUPONT":PRENOM$="JEAN"
     20 PRINT PRENOM$;SPC(3);NOM$
RUN
        JEAN   DUPONT
```

PRINT USING **# . + - ** $$ **$ _ % \\ ! &**

Considérons maintenant l'outil d'édition le plus puissant du BASIC : le 'PRINT USING'

VARIABLES NUMERIQUES : SANS le PRINT USING les VALEURS NUMERIQUES sont CADREES à GAUCHE. Or, c'est généralement à DROITE qu'elles doivent être CADREES.

PRINT USING "#####"; expression numérique

Un format défini par une chaîne de # nous permet de cadrer les nombres à droite : chaque # représente la position d'un chiffre.

```
     10 X=123:Y=1234
     20 PRINT USING "#####";X
     30 PRINT USING "#####";Y
RUN
         123
        1234
```

Sans le PRINT USING, nous aurions obtenu :

```
        123
       1234
```

La partie entre guillemets qui représente le format peut aussi être définie par une variable chaîne de caractères :

```
     10 X=123:Y=1234
     20 FMT$="#####"
     30 PRINT USING FMT$;X
     40 PRINT USING FMT$;Y
RUN
         123
        1234
```

PRINT USING "####.##"; expression numérique

Le nombre de chiffres après la virgule qui doivent être imprimés est précisé dans le format par le nombre de # après le '.'

2 CHIFFRES APRÈS LA VIRGULE

```
10 X=123.456
20 PRINT USING "#####.##";X
RUN
   1234.46
```

On remarque que l'arrondi est assuré automatiquement.

FORMAT MULTIPLE : PRINT USING "format1 format2";Exp1,Exp2

Avec le format multiple, à chaque variable est associé un format :

```
10 X=123.456:Y=67
20 PRINT USING "####.##   ##.##";X,Y
RUN
   123.46   67.00
```

LIBELLES : PRINT USING "libx format1 liby format2";Exp1,Exp2

Des libellés peuvent être insérés dans le format.

```
10 x=1234.567:Y=12
20 PRINT USING "FRANCS #####.##   % ##.##";X,Y
RUN
   FRANCS   1234.57   % 12.00
```

Si le format est le MEME pour plusieurs variables, on fait :

```
10  X=123:Y=345:Z=678
20 PRINT USING "#####";X,Y,Z
RUN
   123  456  678
```

\+ Le signe '+' n'est imprimé que s'il est prévu dans le format :

```
 5 PRINT USING " ####";123     ---->    123
10 PRINT USING "+####";123     ---->   +123
20 PRINT USING "+####";-123    ---->   -123
```

\- Un signe '-' en fin de format provoque l'impression du signe '-' à la fin d'un nombre négatif.

```
10 PRINT USING "####.##-";-15.1   ------->  15.10-
20 PRINT USING "####.##-"; 15.1   ------->  15.10
```

** .2 '*' placés en tête de format provoquent le remplissage par des '*' des positions inoccupées à gauche du nombre imprimé. En outre ces 2'*' spécifient des positions pour 2 chiffres supplémentaires.

```
20 PRINT USING "**####";123    ---->   ***123
30 PRINT USING "**####";1234   ---->   **1234
40 PRINT USING "**####";12345  ---->   *12345
```

$$ .2 '$' en tête de format provoquent l'impression du caractère '$' à gauche du nombre imprimé.

```
10 PRINT USING "$$####.##";456.78 -------->    $456.78
```

$ '$' combine les effets '**' et '$$'.

```
10 PRINT USING "**$####.##";2.34  -------> *****$2.34
```

% Si le nombre de positions spécifié dans le format est insuffisant pour la valeur à imprimer, le message % est imprimé devant la valeur.

```
10 PRINT USING "###";12345     ----> %12345
```

\- Le caractère de soulignement définit le caractère immédiatement suivant comme un libellé. Il permet ainsi d'introduire des caractères de fonction dans les libellés.

```
PRINT USING " _.A= ####";123   ----->    .A=123
```

CHAINES DE CARACTERES

PRINT USING " \ \";expression chaîne

Le nombre d'espaces entre les '\' définit le NOMBRE de CARACTERES-2 à imprimer :

```
10 X$="RENAULT"
20 PRINT USING "\   \";X$
RUN
   RENAU
```

PRINT USING "!";expression chaîne

Imprime le premier caractère d'une expression chaîne.

PRINT "&";expression chaîne

Spécifie une longueur variable de chaîne. La chaîne imprimée est égale à la chaîne spécifiée.

```
10 NOM$="ROULET":PRENOM$="JEAN"
20 PRINT USING " & !";NOM$,PRENOM$
RUN
   ROULET J
```

COMBINAISONS NUMERIQUES / CHAINES

Les formats pour chaînes de caractères et variables numériques sont combinables :

```
10 X$="RENAULT"
20 Y=1234.567:Z=11
30 PRINT USING "Société: \   \ #####.## % ##.##";X$,Y,Z
RUN
   Societe: RENAU  1234.57  % 11.00
```

Programme pour tester PRINT USING

Le programme suivant permet d'entrer par INPUT à la fois le format et le nombre à imprimer.

```
10 INPUT "Format,Nombre ? ";FMT$,NOMBRE
20 PRINT USING FMT$;NOMBRE
30 GOTO 10
RUN
   Format,Nombre ?  ####.##,12.3

   12.30

   Format,Nombre ?
```

POS(0)

Donne la position du curseur (ou de la tête d'impression) dans la ligne courante pour le terminal maître (l'argument 0 n'est pas utilisé).

```
100 PRINT "XXXXX";:X=POS(0):PRINT
110 PRINT TAB(X) "YYYYY"
RUN
   XXXXX
        YYYYY
```

LPOS(0)

Donne la position de la tête d'impression du terminal esclave.

```
100 IF LPOS(0)>80 THEN LPRINT
```

LPRINT

Provoque l'édition sur le terminal esclave. Les règles d'utilisation sont les mêmes que pour PRINT.

***WRITE* expression1,expression2,..** *(version 5.)*

Procède comme pour PRINT ... mais sépare les valeurs par des virgules et imprime les chaînes entre guillemets :

```
10 A=80:B=90:C$="XXXXXX"
20 WRITE A,B,C$
RUN
   80, 90,"XXXXXX"
```

***WIDTH* largeur** *(version 5.)*

Précise le nombre maximum de caractères à imprimer par ligne pour le terminal maître (72 par défaut). Au-delà, un retour-chariot et un saut de ligne sont envoyés par le système BASIC.

La largeur qui peut être spécifiée par une expression entière, doit être comprise entre 15 et 255. Avec 255, les lignes ont une longueur 'infinie', ce qui permet l'envoi de caractères spéciaux (de contrôle) à l'écran ou à l'imprimante sans que BASIC envoie des retour-chariot intempestifs.

Cependant, la valeur donnée par POS est remise à zéro après l'impression de 255 caractères.

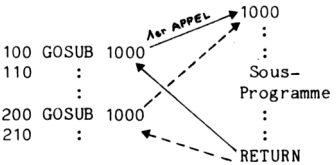
```
100 WIDTH 120
```

***WIDTH LPRINT* largeur**

Pour le terminal esclave.

SOUS-PROGRAMMES GOSUB - RETURN

Plutôt que d'écrire plusieurs fois une même séquence d'instructions en différents endroits d'un programme, on utilise un sous-programme écrit en un endroit unique.

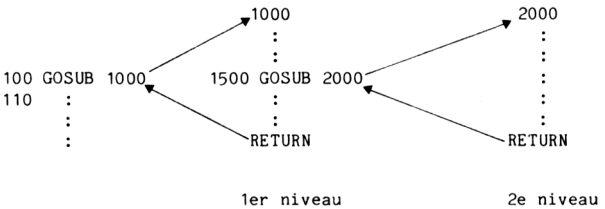


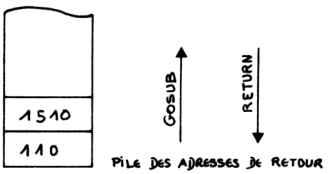
Un sous-programme BASIC s'écrit comme une suite d'instructions mais celle-ci est appelée par une instruction 'GOSUB X' qui provoque un branchement à l'adresse X comme pour un 'GOTO X' mais lorsqu'une instruction 'RETURN' est rencontrée, il y a retour automatique à l'instruction qui suivait l'instruction d'appel 'GOSUB'

Un sous-programme peut lui-même en appeler un autre. A chaque fois qu'une instruction 'GOSUB' est exécutée, l'ADRESSE de RETOUR (adresse de l'instruction qui suit l'instruction d'appel) est stockée dans le haut d'une pile (Dernier Entré Premier Sorti) et à chaque fois qu'une instruction 'RETURN' est exécutée, il y a débranchement à l'adresse de retour du haut de la pile, celle-ci étant alors supprimée de la pile.

Ainsi les retours successifs des sous-programmes se feront bien dans l'ordre.

Il convient, bien sûr, de ne pas entrer dans un sous-programme par un 'GOTO' ou d'en sortir autrement que par un 'RETURN'.





Avec les BASICS interprêtes, une 'entrée' dans un sous-programme par un GOTO ne sera détectée que si la pile des adresses de retour est vide au moment de l'exécution d'une instruction RETURN.

Les sous-programmes sont utilisés non pas seulement pour économiser de la place mémoire, mais aussi (et plutôt) pour les raisons suivantes :

- Par souci de clarté : le 'programme principal' est plus court.

- Les sous-programmes permettent éventuellement de répartir la programmation entre plusieurs programmeurs.

- La mise au point se fait sous-programme par sous-programme. Ainsi on progresse plus sûrement dans la mise au point de l'ensemble du programme.

- Dans le cas où la même séquence d'instructions est répétée, une modification dans cette séquence doit être faite aux différents endroits où elle a été écrite. (fastidieux).

- Lorsqu'un sous-programme est modifié, la 'zone d'intervention' est bien délimitée ; tout le programme ne risque pas d'être remis en cause.

- Les sous-programmes ont aussi l'avantage, lorsqu'en cours de mise au point, il faut 'restructurer' le programme, de permettre une certaine souplesse ; l'ordre d'appel des sous-programmes est à modifier mais ceux-ci ne changent pas.

- Plutôt que d'insérer une nouvelle séquence d'instructions correspondant à l'ajout d'une nouvelle fonction, on peut écrire un sous-programme qui est appelé quand nécessaire.

Les avantages des sous-programmes sont donc multiples. Aussi n'hésitera-t-on pas à en faire un usage abondant, même lorsqu'ils ne comportent que quelques instructions.

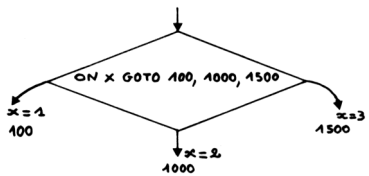
ON .. GOTO .. -ON .. GOSUB ..

***ON X GOTO* N° ligne1, N° ligne2,.....**

(branchement multidirections)

Selon la valeur d'une variable X (ou d'une expression) 1,2,3,.., il y a branchement au N° ligne1, N° ligne2, N° ligne3,...

```
 10 INPUT "Mode? 1,2,3 ";X
 20 ON X GOTO 100,1000,1500
 30 GOTO 10
                 x=1  x=2  x=3

100 .........        ' Traitement 1

1000 .........       ' Traitement 2

1500 .........       ' Traitement 3
```



Si X n'est pas entier, la partie fractionnaire est arrondie. X<0 ou X>255 provoque une erreur
Une valeur de X correspondant à un numéro de ligne non spécifié provoque un branchement à la ligne qui suit ON (ainsi que X=0).

Il est possible d'écrire :

```
10 ON X GOTO 100,1000,1500      'pour X=1,2,3
20 ON X-5 GOTO 200,2500,3000    'pour X=6,7,8
30 PRINT "ERREUR X=";X:STOP
```

ON X GOSUB N° ligne1, N° ligne2, N° ligne3,...

Comme pour ON X GOTO..,..,.., Selon la valeur d'une valeur X (ou d'une expression) 1,2,3..., il y a branchement au *N° ligne 1, N° ligne2, N° ligne3,...*

Mais dès qu'une instruction RETURN est rencontrée, il y a retour à l'instruction qui suit ON X GOSUB.

```
10 ON X GOSUB 100,1000,1500
20 .....                      ' retour en 20
50 STOP


100 ....          ' Sous-Programme 1
110 ....
120 RETURN

1000 ....         ' Sous-Programme 2
1010 ....
1020 RETURN

1500 ....         ' Sous-Programme 3
1510 ....
1520 RETURN
```

Test de conditions au retour d'un sous-programme

Souvent un sous-programme positionne un indicateur que le programme appelant teste au retour. L'utilisation de la fonction de branchement multidirections 'ON Q GOTO ...,..., ...,... ' est dans ce cas très pratique.

```
100 GOSUB 1000:ON Q GOTO 200,300,400

200 TRAITEMENT NORMAL

300 TRAITEMENT ERREUR No 1

400 TRAITEMENT ERREUR No 2
```

```
1000    Sous-programme
              :
              :
1100 Q=1:RETURN    'Tout s'est
                    bien passé
1300 Q=2:RETURN    'Erreur No 1
1400 Q=3:RETURN    'Erreur No 2
```

ON ERROR GOTO .. - ERR - ERL - RESUME .. - ERROR ..
(TRAITEMENT D'ERREURS)

Dès que survient une erreur pendant l'exécution d'un programme et si l'instruction 'ON ERROR GOTO n° ligne' a été prévue, il y a branchement à un programme d'erreur au N° de ligne spécifié dans l'instruction 'ON ERROR'.

Ce programme d'erreur analyse alors l'erreur en testant les valeurs de ERR et ERL qui représentent respectivement le code erreur et le n° de ligne où s'est produite cette erreur.

Après avoir analysé et traité l'erreur, l'instruction RESUME permet au programme d'erreur de provoquer un retour au programme ou s'était produit l'erreur.

```
 10 ON ERROR GOTO 100

 20 INPUT "Diviseur? ";X
 30 PRINT 100/X
 40 GOTO 20

100 IF ERR=11 AND ERL=30 THEN PRINT "Division par ZERO INTERDITE":
        RESUME 20
110 PRINT "Erreur non reconnue":STOP
```

ON ERROR GOTO 0

Ecrit dans un programme de traitement d'erreur, il annule ON ERROR GOTO N° ligne. L'erreur est donc traitée normalement par le système (interruption du programme et message d'erreur)

RESUME

Ecrit à la fin d'un programme de traitement d'erreur, RESUME spécifie où doit se poursuivre l'exécution du programme:

RESUME : OU RESUME 0 :	l'exécution se poursuit au N° de ligne où s'est produit l'erreur
RESUME NEXT :	l'exécution se poursuit au N° de ligne après celui où s'est produit l'erreur
RESUME N° ligne:	permet à l'utilisateur de définir ses propres codes erreurs (compris entre 0 et 255) et de provoquer un branchement à ON ERROR GOTO.. comme si une

ERROR

ERROR N° erreur:	permet à l'utilisateur de définir ses propres codes erreurs (compris entre 0 et 255) et de provoquer un branchement à ON ERROR GOTO.. comme si une erreur avait eu lieu.
10 ERROR 153	provoque un branchement à ON ERROR GOTO ..

Remarque : 'ON ERROR GOTO *N° ligne*' doit être écrit de préférence en tête de programme afin d'être interprétée avant que ne survienne une erreur.

FONCTIONS ARITHMETIQUES

L'argument X peut être une constante, une variable ou une expression numérique.

ABS(X)

Fournit la valeur absolue de X:

100 PRINT ABS(-35) ----> 35

ATN(X)

Donne en radians Arctangente de X (résultat entre -PI/2 et +PI/2)

CDBL(X)

Convertit X en un nombre double précision (avec 16 digits). Elle permet aussi d'effectuer une opération en double précision :

```
100 FOR I%=1 TO 10 : PRINT 1/CDBL(I%) :NEXT I%
```

CINT(X)

Convertit X en un entier avec arrondi :

```
100 PRINT CINT( 1.6)      ------->     1
110 PRINT CINT(-1.2)      ------->    -2
```

(X doit être compris entre -32768 et 32767)

COS(X)

Donne le Cosinus de X exprimé en radians.

CSNG(X)

Convertit X en un nombre simple précision (6 digits) :

```
100 X= 1234.56789123:A=CNSG(X=):PRINT A ----> 1234.57
```

EXP(X)

Donne l'exponentielle de X.

FIX(X)

Supprime les chiffres après la virgule :

```
100 PRINT FIX( 2.2)     ----->      2
110 PRINT FIX(-2.2)     ----->     -2
```

FRE(0)

Donne la place libre en mémoire centrale (l'argument 0 n'est pas utilisé)

INT(X)

Donne la partie entière de X avec arrondi

```
100 PRINT INT( 2.2)     ----->      2
110 PRINT INT(-2.2)     ----->     -3
```

LOG(X)

Fournit le logarithme (naturel) de X

RANDOMIZE

(n'est pas une fonction) Sans cette instruction, la séquence des nombres aléatoires est toujours la même.

A l'exécution du programme, cette instruction envoie un message à l'opérateur pour lui demander un nombre qui permet au générateur aléatoire de générer une séquence aléatoire non standard.

```
 10 RANDOMIZE
 20 FOR I=1 TO 5:PRINT RND(1);:NEXT I
RUN
    Random Number Seed (0-65529) ? 3  (3 est la reponse de
                                       l'opérateur)
    .88598 .484668 .586328 .119426  .709225
```

RND(X)

```
X>0 ou sans X        ------> nombre aléatoire suivant.
X=0                  ------> meme nombre aléatoire
X<0                  ------> réinitialise la séquence

Ex:     FOR I=1 TO  5:PRINT INT(RND(1)*100);:NEXT I

   RUN
          24 30 31 51 1
```

Sur TRS80: pour X=>0 et X<1 -----> nombre aléatoire entre 0 et 1
X>1 et X<32767 ---> nombre aléatoire entier entre 0 et 32767
(c'est la partie entière de l'argument qui est considérée)

SGN(X)

Donne le signe X : on obtient respectivement -1,0,+1 pour les valeurs négatives, nulles, positives.

SIN(X)

Donne le Sinus de X exprimé en radians.

SQR(X)

Donne la racine carrée de X. (X doit être positif)

TAN(X)

Donne la tangente de X.

DEFINITION DE FONCTIONS DEF FN

En plus des fonctions internes (telles que SQR, SGN, INT, etc.), l'utilisateur peut définir ses propres fonctions par :

DEF FNXX(X,Y,Z,..)=EXPRESSION(X,Y,Z,..)

où XX représente un nom choisi par l'utilisateur pour identifier sa fonction (les règles d'appelation sont les mêmes que pour les variables) et X,Y,Z,.. les arguments de la fonction.

(sur certaines versions de BASIC, le nombre d'arguments est limité à 2).

Plus tard, cette fonction sera appelée par le programme avec les valeurs réelles des paramètres.

Exemples

Fonction d'arrondi

```
10 DEF FNAR(X)=INT(X+.5)
20 A=123.6
25 B=345.2
30 PRINT FNAR(A)
40 PRINT FNAR(B)

RUN
     124
     345
```

Fonction maximum de 2 nombres

```
10 DEF FNMX(X,Y)=-((X>Y)*X+(X<=Y)*Y))
20 A=15:B=20          └──────┘ s'évalue en 0 ou -1
30 PRINT FNMX(A,B)
RUN
    20
```

Fonction d'arrondi avec décimales

```
10 DEF FNAR(X)=INT(X*100+.5)/100
20 PRINT FNAR(123.456) ---> 123.46
```

Remplissage par des caractères à gauche

```
10 DEF FNX$(X,RP$)=RIGHT$(STRING$(6,RP$)+RIGHT$(STR$(X),LEN(STR$(X))-1),6)

20 PRINT FNX$(7,"0") ----->   0007
30 PRINT FNX$(7,"@") ----->   @@@7
```

Une fonction ne peut être écrite que sur une seule ligne de 255 caractères au plus.

- C'est de préférence en tête de programme que sont écrites les fonctions de façon à être interprétées avant qu'elles ne soient appelées.

- Une même fonction (même nom) peut être réinterprêtée différemment en cours d'exécution.

SWAP élément1,élément2 : (version 5.)

Les valeurs de *Elément 1* et de *Elément 2* sont échangées. Ces éléments doivent être des variables ou des éléments de tableaux du même type.

```
100 A(1)=10:A(2)=20
110 PRINT A(1),A(2)
120 SWAP  A(1),A(2)
130 PRINT A(1),A(2)
RUN
     10   20
     20   10
```

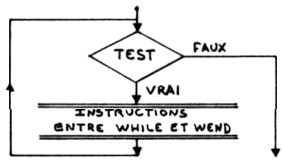
WHILE ... WEND (version 5.)

```
WHILE expression logique
        :
        :
WEND
```

exécute toutes les instructions entre WHILE et WEND tant que l'expression logique est vraie (différent de 0). Dès que l'expression devient fausse (=0) le programme se poursuit après WEND.

En fait, l'expression spécifiée dans WHILE peut aussi être arithmétique.

```
10 INVERSION=1

20 WHILE INVERSION><0

30     INVERSION=0

40     FOR I=1 TO NELEMENT-1
50       IF A$(I+1)<A$(I) THEN SWAP A$(I),A$(I+1):INVERSION=1
60     NEXT I

70 WEND
```



L'équivalent de WHILE avec la boucle FOR serait :

```
10 FOR I=1 TO INFINI
20   IF expression=0 THEN GOTO SORTIE
       .
       .
30 NEXT I

SORTIE: .......
```

CHAIN - COMMON : (MICROSOFT 5.) (APPEL DE PROGRAMMES)

CHAIN MERGE nom-programme, n° ligne de branchement, ***ALL, DELETE*** ligne1-ligne2

Appelle un programme en passant les variables du programme courant.

```
100 CHAIN "PAYE"
```

Le n° de ligne de branchement (qui peut être une expression) représente la ligne où l'exécution du programme appelé commencera.

```
100 CHAIN "PAYE", 1000
```

Si ALL est spécifié, toutes les variables du programmes appelant sont passées au programme appelé.

L'option MERGE permet d'appeler un sous-programme qui "recouvre" une partie du programme appelant. Le sous-programme appelé doit être sous forme ASCII (voir SAVE et MERGE).

```
CHAIN MERGE "PAYE",1000
```

Un partie du programme peut également être supprimée par DELETE.

```
CHAIN "PAYE",1000,DELETE 2000-2500
```

(la version compilée de 5. ne supporte pas ALL,MERGE et DELETE)

COMMON **liste de variables**

Utilisé conjointement avec CHAIN, COMMON permet de passer certaines variables au programme appelé.

```
100 COMMON A,B,C,D(),G$
110 CHAIN "PAYE",10
```

Bien que COMMON puisse apparaître à un endroit quelconque du programme, il est préférable de l'écrire en tête de programme pour des raisons de clarté.

Une même variable ne doit pas apparaître dans différentes instructions COMMON.

Les tableaux sont spécifiés par () à la suite du nom de tableau.

Si toutes les variables doivent être passées, il faut utiliser l'option ALL de CHAIN

PEEK - POKE : (ACCES DIRECT EN MEMOIRE CENTRALE)

PEEK et POKE permettent d'accéder directement à la mémoire centrale en lecture et en écriture.

***PEEK* (adresse)**

Cette fonction fournit le contenu de l'adresse mémoire spécifiée (entre 0 et 65536).

Le résultat qui représente le contenu d'un octet est compris entre 0 et 255.

```
100 A=PEEK(1000)
110 PRINT HEX$(PEEK(1000))
120 PRINT HEX$(PEEK(&H3E8))
```

***POKE* adresse, octet**

Range à l'adresse mémoire spécifiée la valeur de l'octet (comprise entre 0 et 255)

```
200 POKE 1000,122
```

INP - OUT - WAIT (ENTREES-SORTIES)

***INP* (numéro de port)**

Lit un octet sur le 'port' spécifié. Le n° de port doit être compris entre 0 et 255.

```
100 A=INP(255)
```

***OUT* numéro de port, données**

Envoie un octet sur le 'port' spécifié. 'no port' et 'données' doivent être compris entre 0 et 255. Ils peuvent être des expressions entières.

```
200 OUT 32,100
```

***WAIT* numéro de port, masque, sélection**

Suspend l'exécution du programme jusqu'à ce que l'entrée d'un port ait un bit particulier positionné à 1.

- Si 'sélection' est nul, un 'ET' effectué entre l'entrée du port et 'masque' fournit un résultat qui, s'il est égal à zéro, provoque un retour à l'instruction WAIT qui s'exécute à nouveau.

- Si au contraire, le résultat est différent de zéro, l'exécution de WAIT se termine. L'exécution du programme se poursuit.

Pour 'sélection' différent de zéro, un 'OU exclusif' est d'abord effectué entre l'entrée du port et 'sélection' puis un 'ET' est effectué entre le résultat obtenu et 'masque'.

10 WAIT 22,4	suspend l'exécution jusqu'à ce que le troisième bit (2 puissance 2) du port 22 soit positionné à 1
10 WAIT 22,7	suspend l'exécution jusqu'à ce que l'un des 3 premiers bits soit positionné à un.

```
10 '                IMPRESSION D'UN TABLEAU 2 DIMENSIONS
20 '                ====================================
30 '
40 DATA "FLINS   ","CLEON   ",BILLANCOURT,"SANDOUVILLE","DOUAI  ","ORLEANS":
                              FOR I=1 TO 6:READ USINE$(I):NEXT I
50 DATA R5,R6,R12,R18,R20,R30,R40:FOR I=1 TO 7:READ TYPE$(I):NEXT I
60 DIM STOCK(7,6)  '                                7 types et 6 usines
70 '
80 '  Saisie Tableau
90    STOCK(1,1)=52:STOCK(1,2)=79:STOCK(1,3)=87:STOCK(2,2)=99:STOCK(2,5)=119
100 '-----------------------------------------------------------------------
110 '                                            Impression titre USINES
120 FOR I=1 TO 5
130    LPRINT TAB(15+10)
140 '
150    FOR J=1 TO 6
160      X$=MID$(USINE$(J),I,1):LPRINT X$;"    ";
170    NEXT J
180 '
190    LPRINT
200 NEXT I
210 '
220 LPRINT
230 '-----------------------------------------------------------------------
240 '                                            Impression du tableau STOCK
250 FOR TYPE=1 TO 7
260    LPRINT TAB(15) TYPE$(TYPE); TAB(15+6)
270    TTYPE=0
280 '
290    FOR USINE=1 TO 6
300       IF STOCK(TYPE,USINE)><0 THEN LPRINT USING "#####";STOCK(TYPE,USINE);
             ELSE   LPRINT "    .";
310       TTYPE=TTYPE+STOCK(TYPE,USINE)
320       CUMUL(USINE)=CUMUL(USINE)+STOCK(TYPE,USINE)
330    NEXT USINE
340 '
350    LPRINT USING "#####";TTYPE
360 NEXT TYPE
370 '
380 LPRINT:LPRINT TAB(15+6)
390 '
400 FOR USINE=1 TO 6
410   LPRINT USING "#####";CUMUL(USINE);
420 NEXT USINE
430 '
```

TABLE USINE$

USINE$ (1)	F	L	I	N	S
USINE$ (2)	C	L	E	O	N
USINE$ (3)	B	I	L	L	A
	S	A	N	D	O
	D	O	U	A	I
USINE$ (6)	O	R	L	E	A

5

	F L I N S	C L E O N	B I L L A	S A N D O	D O U A I	O R L E A	
R5	52	79	87	.	.	.	218
R6	.	99	.	.	119	.	218
R12	.	.	.	.	.	.	0
R18	.	.	.	.	.	.	0
R20	.	.	.	.	.	.	0
R30	.	.	.	.	.	.	0
R40	.	.	.	.	.	.	0
	52	178	87	0	119	0	

```
10 '
20 '              RECHERCHE DICHOTOMIQUE      (insertion+éléments égaux)
30 '              =====================
40 '
50 '       .La dichotomie consiste,lorsqu'on recherche un élément dans une
60 '        suite ordonnée ,à le comparer à l'élément du milieu de cette
70 '        suite afin de SELECTER la MOITIE où il se trouve.
80 '       .On procède de la même façon sur la moitié sélectée et après
90 '        quelques comparaisons l'élément cherché est retrouvé.
100 '       (s'il n'existe pas ,on obtient la position d'insertion)
110 '       (Si plusieurs éléments sont égaux ,ce programme donne le 1er )
120 '
130 '              INFERIEUR --->  [ 2 ]
140 '                              [ 4 ]
150 '  Elément cherché [4]         [ 6 ]
160 '                              [   ]
170 '
180 '             MILIEU  ------>  [ 1Ø ]
190 '                              [   ]
200 '
210 '
220 '
230 '             SUPERIEUR ---->  [ 2Ø ]
240 '
250 '
260 TAILLE=10
270 FOR I=1 TO TAILLE:A(I)=2*I:NEXT I                  ' Remplissage d'une table
280 '
290 INPUT "Nombre cherché? ";NCHERCHE
300 '
310 GOSUB 370:ON Q GOTO 320,330                        ' Résultat ---> NLIGNE
320 PRINT "Elément trouvé en :";NLIGNE:GOTO 290        ' Elément trouvé: Q=1
330 PRINT "Elément  non trouvé";NLIGNE:GOTO 290        ' Elément non trouvé:Q=2
340 '
350 '-----------------------------------------------------------------------
360 '
370 INFERIEUR=1:SUPERIEUR=TAILLE
380 '
390 IF INFERIEUR>SUPERIEUR THEN  Q=2:NLIGNE=MILIEU:IF NCHERCHE>A(NLIGNE) THEN
          NLIGNE=NLIGNE+1:RETURN ELSE RETURN            ' Elément non trouvé
400 MILIEU=INT((INFERIEUR+SUPERIEUR)/2)
410 '
420 IF NCHERCHE=A(MILIEU) THEN
     IF A(MILIEU-1)><A(MILIEU) THEN Q=1:NLIGNE=MILIEU:RETURN
             ELSE SUPERIEUR=MILIEU-1:GOTO 390          ' Elément trouvé
430 '
440 IF NCHERCHE<A(MILIEU) THEN SUPERIEUR=MILIEU-1:GOTO 390
450 IF NCHERCHE>A(MILIEU) THEN INFERIEUR=MILIEU+1:GOTO 390
460 '
470 ' run
480 '
490 '    Nombre cherche?    4
500 '    Elément trouvé en: 2
510 '
520 '    Nombre cherché?    5
530 '    Elément non trouvé:3
```

```
10 '                    RECHERCHE AUTOMATIQUE D'ECHELLE pour HISTOGRAMME
20 '                    ================================================
30 '
40 '  Ce programme permet d'obtenir un histogramme sans que l'échelle soit
50 '  prévue à l'avance.
60 '
70 DIM TX(20)
80 '
90 ' Remplissage d'une table par des nombres aléatoires
                 pour tester le programme
100 '
110 FOR I=1 TO 15:TX(I)=INT(RND(1)*20):LPRINT USING "###";TX(I);:NEXT I
120 '-----------------------------------------------------------------------
130 FOR I=1 TO 15                                      ' Recherche du plus grand
140   IF TX(I)>PLUSGRAND THEN PLUSGRAND=TX(I)
150 NEXT I
160 '
170 ECH= 10/PLUSGRAND                                          ' Taille max=10
180 ECH=INT(ECH*8)/8                                           ' Ech mini .125
190 '-----------------------------------------------------------------------
200 '                                                          ' Edition
210 LPRINT:LPRINT: LPRINT "Histogramme ":LPRINT ""
220 '
230 FOR J=10 TO 1 STEP-1
240 '
250     FOR I=1 TO 15
260       IF TX(I)*ECH=>J THEN LPRINT "  *"; ELSE LPRINT "   ";
270     NEXT I
280 '
290     LPRINT ""
300 NEXT J
310 '
320 LPRINT "":FOR I=1 TO 15:LPRINT USING "###";I;:NEXT I:
            LPRINT "   Echelle:";ECH
330 END
340 '
```

```
  4  6  6 10  1 15  9  7 19 18 14  0 19  0 19

Histogramme

                          *  *        *     *
                          *  *        *     *
                 *        *  *  *     *     *
                 *        *  *  *     *     *
          *      *        *  *  *     *     *
          *      *  *     *  *  *     *     *
     *  *  *     *  *  *  *  *  *     *     *
  *  *  *  *     *  *  *  *  *  *     *     *
  *  *  *  *     *  *  *  *  *  *     *     *

  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15   Echelle: .5
```

```
10 '       TRI par la METHODE de SHELL
20 '       ==========================
30 '
40 '     .La comparaison ne s'effectue pas entre 2 nombres consécutifs comme
50 '      pour 'ripple sort' mais entre 2 nombres séparés par un 'pas' égal au
60 '      départ à la moitié de la taille de la table à trier.
70 '     .Ce pas est successivement divisé par 2.
80 '     .Le tri s'achève avec un pas de 1 (comme pour 'ripple sort').
90 '
100 '    .Dès que le nombre de nombres à trier devient supérieur à 50, cette
110 '     méthode est plus rapide que celle du 'ripple'.
120 '
130 INPUT "Taille? ";TAILLE
140 DIM NOMB(TAILLE)
150 '
160 FOR I=1 TO TAILLE:NOMB(I)=RND(1):NEXT I   ' Remplissage d'une table
170 '                                          de nombres aléatoires
180 GOSUB 280:FOR I=1 TO TAILLE:PRINT NOMB(I):NEXT I
190 LPRINT "Taille:";TAILLE; TAB(20)"  Il y a eu :";C;
            TAB(40)"Comparaisons et ";V;"Inversions"
200 END
210 '-----------------------------------------------------------------------
220 '
230 '                 9 8 7 6 5 4 3 2 1 0
240 '                 -         -
250 '                 ¦         ¦
260 '                   Inverser
270 '
280 PAS=TAILLE
290 '
300 PAS=INT(PAS/2):IF PAS<1 THEN RETURN
310 INVERSION=0
320 '
330    FOR I=1 TO TAILLE-PAS
340      C=C+1                                  ' Compteur de comparaisons
350      N=I+PAS:IF NOMB(I)>NOMB(N) THEN
                 SWAP NOMB(I),NOMB(N):INVERSION=1:V=V+1
360    NEXT I
370 '
380 IF INVERSION=1 THEN GOTO 310 ELSE GOTO 300      ' On recommence
390 '                                                 jusqu'à  ce qu'il
400 '                                                 n'y ait plus d'inversion
410 '-----------------------------------------------------------------------
420 '
430 '                   SHELL                          RIPPLE
440 '
450 '  Taille ¦   Comparaisons   Inversions  ¦  Comparaisons  Inversions
460 '         ¦                              ¦
470 '   10    ¦     47            8         ¦     45           19
480 '   20    ¦    230           39         ¦    190          104
490 '   50    ¦   1010          180         ¦   1170          642
500 '  100    ¦   3382          515         ¦   4914         2657
510 '  200    ¦   6791         1035         ¦  19669        10541
```

IIème Partie

Les fichiers en BASIC Microsoft

CHAPITRE 2-1

FICHIERS A ACCES DIRECT (random)

```
OPEN
FIELD #
PUT # LSET RSET.GET#
MKI$ MKD$ -CVI CVS CVD
CLOSE
LOC LOF
```

Les fichiers permettent de mémoriser des informations qui peuvent être ensuite consultées ou modifiées. Les supports de ces fichiers sont généralement magnétiques (bande ou disque).

Une disquette est composée de plusieurs pistes (35 pour les minidisquettes de 5 pouces 1/2 ou 77 pour disquettes 8 pouces), chacune de ces pistes étant découpée en plusieurs "secteurs" (10 pour les minidisquettes ou 26 pour les disquettes).

Chaque secteur (128 ou 256 caractères) a une "adresse physique" :

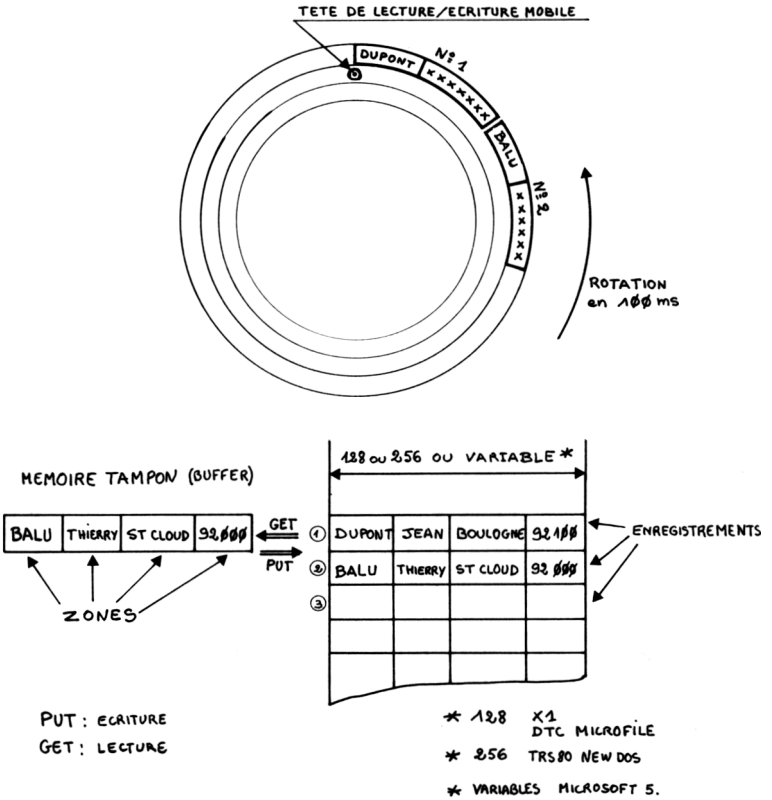
(Numéro piste, Numéro secteur à l'intérieur de la piste)

L'accès à un secteur se fait d'abord en positionnant directement la tête de lecture/écriture sur la piste cherchée puis en recherchant le secteur à l'intérieur de la piste.

On parle dans ce cas d'accès et d'organisation directs, bien que la recherche à l'intérieur d'une piste soit séquentielle. Le programmeur ne connaît qu'un numéro d'ordre d'enregistrement à l'intérieur d'un fichier. C'est le logiciel-système qui assure la correspondance avec l'adresse physique d'un secteur par un calcul.

Les anciennes versions de BASIC ne proposaient que des enregistrements de longueur fixe (128 ou 256 c.). Maintenant, le programmeur peut définir la longueur des enregistrements qui est la même pour tous ceux d'un fichier.

L'avantage des fichiers à accès direct sur les fichiers séquentiels est de permettre des consultations et des mises à jour immédiates (en "temps réel").



FICHIER A ACCES DIRECT

OPEN

L'accès aux informations d'un fichier ne peut se faire que s'il est ouvert. L'instruction OPEN, en ouvrant un fichier, lui affecte une mémoire tampon (buffer) où transiteront les informations au cours des opérations de lecture et d'écriture dans ce fichier.

OPEN "R",#*no fichier*,"U:*nom fichier*",*longueur enregistrement*

←-MICROSOFT 5. (ou U=A,B,C,....) représente l'unité de disquette (A par défaut)

OPEN "R",#*no fichier*,"*nom fichier : no unité*"

←-sur TRS80 NEWDOS (no unité=0 par défaut)

'R' spécifie le type de fichier (RANDOM). Ce type peut être spécifié par une expression du type chaîne.

'no fichier' qui représente en fait le numéro du buffer attribué au fichier ouvert, servira dans la suite du programme à référencer ce fichier.

'U:*nom fichier*' peut aussi être spécifié par une expression du type chaîne ("U:"+NF$ par ex)

'longueur enregistrement' qui peut être définie par une expression entière, est de 128 par défaut pour la version 5.

```
A   10 OPEN "R",#1,"CLIENT",200

B   10 INPUT "Nom de fichier? ";NF$
    20 OPEN "R",#1,NF$,64
```

L'instruction OPEN de l'exemple A ouvre le fichier "CLIENT" sous le n° 1 (lui attribue le buffer n° 1) et affecte une longueur de 200 à ses enregistrements.

Avec Microsoft 5. la longueur maxi des enregistrements (128 par défaut) et le nombre de fichiers ouverts maxi (3 par défaut) sont définis au moment de l'appel de BASIC :

>MBASIC /S:*Longueur maxi enregistrement* /F:*Nombre de fichiers*

Un même fichier peut être ouvert avec des numéros différents. Dans ce cas il lui est affecté un BUFFER par numéro. (Voir ouverture multiple).

FIELD #No fichier, longueur1 AS var1$,longueur2 AS var2$,.... (DESCRIPTION DES ZONES)

Un fichier à accès direct est composé d'enregistrements référencés par des numéros. Chacun de ces enregistrements est lui-même compose de zones dont les longueurs sont définies par une instruction FIELD (champ) :

FIELD #*No fichier,longueur1* AS *var1$,longueur2* AS *var2$,....*

Cette instruction FIELD positionne les zones à l'intérieur d'un enregistrement dans l'ordre de leur description en leur réservant les longueurs spécifiées. Elle n'est généralement programmée qu'une fois, après l'OPEN par exemple.

```
20 FIELD #1,15 AS NOM$,10 AS PRENOM$,25 AS RUE$,15 AS VILLE$,4 AS CPOST$
```

15 caractères sont réservés pour le nom
10 caractères sont réservés pour le prénom
25 caractères sont réservés pour la rue
15 caractères sont réservés pour la ville
5 caractères sont réservés pour le code postal.

15	10	25	4	
DUPONT	JEAN	73 rue de SEVRES	92109	
NOM$	PRENOM$	RUE$	CPOST$	

Le programmeur n'a accès, pour un fichier et par buffer, qu'à un seul enregistrement à la fois.

PUT#No fichier,No enregistrement (Ecriture) LSET - RSET

Avant d'écrire dans un enregistrement par une instruction 'PUT#*no fichier, no enregistrement*', on affecte aux différentes zones de la mémoire tampon du fichier (buffer) les différentes valeurs qu'elles doivent avoir par l'instruction LSET :

LSET *variable définie dans field=expression chaîne de caractères*

```
 10 INPUT "Nom? ";X$
 20 LSET NOM$=X$                    ' Transfere X$ dans NOM$

 30 INPUT "Prenom? ";X$
 40 LSET PREN$=X$
      .
      .
100 PUT #1,no enregistrement        ' Ecriture dans le fichier
```

LSET signifie que la valeur est cadrée à gauche dans la zone (RSET cadre à droite). Les positions non occupées dans la zone sont complétées par des espaces (codes ASCII=32) : par ex. LSET NØ="DUPONT" range 'DUPONT' dans NØ en complétant par 9=(15-6) espaces à droite si la longueur de NØ prévue est de 15.

Attention : Les variables définies dans le FIELD ne sont pas utilisables comme les autres variables : ne pas faire INPUT "Nom?";NOMØ si NOMØ a été définie dans FIELD.

Si des zones ne se voient pas attribuer de valeur à la saisie, il convient de procéder à leur initialisation. Sinon elles prendraient, pour l'enregistrement créé, les valeurs présentes dans le buffer au moment de la création. (C'est-à-dire celles de l'enregistrement précédemment lu ou écrit).

Uniquement pour enregistrement de longueur fixe (128 ou 256) : (pour MICROSOFT 5., voir LOF - LOC)

Le plus simple est de lire avant la saisie un enregistrement qui n'a pas encore été créé : GET#1,LOF(1)+1. Le buffer est ainsi initialisé avec des valeurs nulles (codes ASCII=0)

```
  5 GET #1,LOF(1)+1        ' Initialisation du buffer
                           ' avec des valeurs ASCII NULLES
 10 INPUT "Nom? ";X$
 20 LSET NOM$=X$
      .
      .
100 PUT #1,NO ENREGISTREMENT
```

Attention : Sur TRS80, si l'enregistrement N°100 par ex. est créé dans un fichier vide, les 99 enregistrements précédents sont eux aussi créés avec des valeurs quelconques à l'intérieur.

Numéros d'enregistrements

```
MICROSOFT 5.      1 à 32767
TRS80 .......... 1 à 355
DTC MICROFILE .. 0 à 4095
```

Si le n° d'enregistrement n'est pas précisé dans l'instruction PUT, il y a écriture dans l'enregistrement immédiatement après l'enregistrement courant.

GET# N° fichier, N° enregistrement (LECTURE)

La lecture d'un enregistrement en mémoire centrale (dans le buffer prévu) se fait par :

GET # *N° Fichier,N° enregistrement*

Toutes les valeurs des différentes zones de l'enregistrement lu sont alors disponibles dans le buffer. On peut faire par exemple :

```
PRINT NOM$,PREN$,RUE$
```

Si le numéro d'enregistrement n'est pas précisé dans GET, c'est l'enregistrement immédiatement après l'enregistrement courant qui est lu.

MODIFICATION DE ZONES

Pour modifier une zone d'un enregistrement, celui-ci doit d'abord être lu en mémoire centrale. La nouvelle valeur est ensuite affectée à la zone et enfin l'enregistrement est réécrit.

```
10 GET #1,X              ' Lecture en mémoire centrale
20 LSET VILLE$="PARIS"   ' Modification d'une ZONE
30 PUT #1,X              ' Réécriture dans le fichier
```

TYPES DE ZONES NUMERIQUES : MKI$ - MKS$ - MKD$ - CVI - CVS - CVD

Ne peuvent être définies dans un 'FIELD' que des chaînes de caractères. Les variables numériques doivent être transformées et compactées sous forme de chaînes de caractères de la façon suivante :

LSET *variable définie dans field*=MKx$ (expression numérique)

avec x=I pour variable entière (integer)
x=S pour variable simple précision
x=D pour variable double précision

```
10 INPUT "Code Postal? ";CPOST
20 LSET CPOST$=MKS$(CPOST)     ' Si CPOST est un nombre en simple
                               ' précision
30 PUT #1,X
```

A la lecture la conversion inverse sera réalisée par CVx (variable définie dans field) :

```
50 GET #1,X
60 CPOST=CVS(CPOST$)
70 PRINT CPOST
```

Selon les valeurs numériques, les fonctions de conversions et les réservations dans le FIELD sont :

1 --> 256	LSET X$=CHR$(X)	1 CARACT	X=ASC(X$)
-32768 --> +32767	LSET X$=MKI$(X)	2 CARACT	X=CVI(X$)
simple precision	LSET X$=MKS$(X)	4 CARACT	X=CVS(X$)
double precision	LSET X$=MKD$(X)	8 CARACT	X=CVD(X$)
	ECRITURE	RESERVATION dans FIELD #	LECTURE

Définition de structures multiples

Plusieurs structures pour un même enregistrement peuvent être définies simultanément :

```
10 FIELD #1,15 AS NOM$,  10 AS PREN$, 25 AS RUE$,15 AS VILLE$,...
20 FIELD #1,15 AS ZN$(1),10 AS ZN$(2),25 AS ZN$(3),.....
30 FIELD #1,25 AS NP$                , 40 AS ADR$            ,...
```

- Le NOM est connu à la fois sous les noms de 'NOMØ' et de 'ZNØ(1)' Ainsi, on accède aux différentes zones soit par un indice (lors d'une saisie par ex.) soit par un nom propre (pour une édition).

- NPØ représente la réunion du NOM et du PRENOM,ADRØ la réunion de la RUE et de la VILLE.

(L'instruction FIELD définissant des pointeurs internes vers les adresses des zones du buffer, plusieurs pointeurs peuvent être définis vers une même adresse).

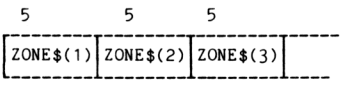
Rien n'empêche de faire chevaucher une table sur 2 buffers.

Avec la version 5., une instruction FIELD écrite sans ouverture de fichier permet de définir une structure sur une chaîne de caractères.

Ecriture condensée

Si tous les éléments d'une table ont la même longueur, la définition des zones se condense par exemple de la façon suivante :

5 caractères sont réservés pour chaque zone.

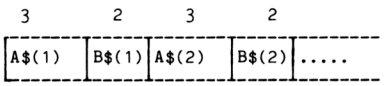



```
10 FOR I=1 TO 10
20   FIELD #1,(I-1)*5 AS D$,5 AS ZONE$(I)
30 NEXT I
```

La variable DØ n'est utilisée que pour positionner les différentes zones dans le buffer.

Définition de plusieurs tables

2 tables AØ et BØ sont définies.




```
10 FOR I=1 TO 10
20   FIELD #1,(I-1)*5 AS D$,3 AS A$(I),2 AS B$(I)
30 NEXT I
```

Enregistrements logiques

Le découpage d'une enregistrement physique en plusieurs enregistrements logiques, lorsque la longueur des enregistre-

ments n'a pu être choisie, peut être fait par exemple de la manière suivante :

DUPONT	xxxxxxxxxxxxxxx	DURAND	xxxxxxxxxxxxxxxx	MARTIN	xxxxxxxxxxxxxx
Position No 1		No 2		No 3	

Si NLOGIQUE est le numéro d'enregistement logique, le n° d'enregistrement physique et la position de l'enregistrement logique dans celui-ci sont donnés par :

```
NPHYSIQUE=INT(NLOGIQUE/3)+1    ' 3 enregistrements logiques
POSITION =NLOGIQUE MOD 3       ' par enregistrement physique
```

NLOGIQUE=8 --> NPHYSIQUE=INT(8/3)+1=3 POSITION=8 MOD 3=2

La définition des variables à l'intérieur d'un enregistrement logique peut être faite dynamiquement par :

```
500 POSITION= ( 1,2 ou 3) :GOSUB 1000    ' POSITION:position dans
                                         ' l'enregistrement physique
510 GET #1,X
520 LSET NOM$="DUPONT"
530 PUT #1,X
540 STOP

1000 FIELD #1,(POSITION-1)*85 AS D$,12 AS NOM$,10 AS PREN$,.......:RETURN
```

Chaque nom de variable est affecté à l'un des 3 enregistrements logiques possibles. L'instruction FIELD# doit bien sûr être exécutée à chaque fois que l'on change d'enregistrement logique.

ATTENTION : L'instruction FIELD est longue à exécuter si beaucoup de variables y sont définies.

Autre solution : Seules sont définies dans le buffer, les zones du premier enregistrement logique. Pour effectuer une mise à jour sur le second enregistrement logique, il faut, après lecture de l'enregistrement physique, sauvegarder le premier enregistrement logique, transférer la seconde moitié du buffer dans la première, effectuer les mises à jour, transférer la première moitié du buffer dans la seconde et enfin, restituer à la première moitié du buffer ses valeurs initiales. L'enregistrement physique peut alors être réécrit.

```
 10 FIELD #1,12 as NOM$,10 as PREN$,.....
 20 FIELD #1,128 as E1$,128 as E2$       ' 2 enregistrements logiques

 30 POSITION=1 ou 2 :GET #1,X
 40 IF POSITION=2 THEN SAUV$=E1$:LSET E1$=E2$
 50 LSET NOM$="DUPONT"

 60 IF POSITION=2 THEN LSET E2$=E1$:LSET E1$=SAUV$
100 PUT #1,X
```

Ouvertures multiples d'un même fichier

L'ouverture d'un fichier sous plusieurs numéros permet de lui attribuer plusieurs buffers et ainsi de disposer éventuellement d'autant d'enregistrements du même fichier en mémoire centrale.

Lorsqu'elle existe, c'est avec prudence que l'on profitera de cette possibilité. On n'utilisera qu'un seul N° de fichier pour les CREATIONS, l'autre ne servant que pour les consultations ou modifications d'enregistrements déjà existants. Ce dernier numéro de fichier ne sera pas clos (le bloc de contrôle à jour serait écrasé).

En 5. un même fichier peut être ouvert indifféremment en accès direct ou en séquentiel et simultanément sous des numéros différents.

CLOSE#N°FICHIER,#N°FICHIER,...

Clot le fichier spécifié et libère le buffer qui lui était alloué.

Un bloc de contrôle pour chaque fichier, amené en mémoire centrale au moment de l'OPEN, qui contient notamment des pointeurs vers les différents enregistrements du fichier est bien sûr mis à jour en mémoire centrale à chaque création d'enregistrement.

Mais sur certains systèmes (tels que le TRS80), c'est seulement la fermeture du fichier qui provoque la sauvegarde sur disque de ce bloc de contrôle. On prendra donc soin, sur TRS80, de clore les fichiers en fin de session.

Une méthode, pour s'affranchir de cette contrainte, consiste à créer l'enregistrement de N° le plus élevé puis à clore le fichier. Par la suite, les pointeurs (systèmes) vers les enregistrements ayant été créés, la fermeture des fichiers ne sera plus nécessaire. Mais dans ce cas, l'avantage de l'allocation dynamique de la place sur disque est, bien entendu, perdu.

```
CLOSE #1,#2
```

END et NEW ferment les fichiers.

LOF - LOC : (POUR ENREGISTREMENTS DE LONGUEUR FIXE 128 OU 256)

***LOF* (N° fichier)**

Permet de connaître le N° du dernier enregistrement dans le fichier (le n° le plus élevé) et ainsi d'allouer un enregistrement lors d'une création.

```
100 ARANG=LOF(1)+1
110 GET #1,ARANG
120 LSET N$="DUPONT"
        .
        .
200 PUT #1,ARANG
```

Selon les BASICS, LOF représente le numéro de l'enregistrement occupé de rang le plus élevé ou ce numéro+1 (DTC MICROFILE).

***LOC* (N° fichier)**

Représente le N° du dernier enregistrement+1 qui a été référencé (en écriture ou lecture)

Remarque : Les instructions d'entrées/sorties avec les fichiers sont exécutables en mode direct. Ceci est utile au moment de la mise au point.

```
GET #1,3
PRINT N$

DUPONT
OK

LSET N$="XXXX"
PUT #1,3
```

LOF* (N° fichier) : *MICROSOFT 5.

LOF indique le nombre de blocs de 128 octets utilisés dans l'extent le plus récemment référencé en lecture ou en écriture (1 extent=128*128 octets). Un bloc de 128 partiellement occupé est compté comme utilisé.

Avec des enregistrements de longueur 128 et pour un fichier de taille inférieure à un extent, LOF représente bien le nombre d'enregistrements utilisés, mais seulement dans ce cas.

Comment donc allouer un enregistrement au moment d'une création ?

- On peut gérer soi-même un compteur d'enregistrements occupés, sauvegardé dans l'enregistrement N° 1 par exemple :

```
10 OPEN "R",1,"XX",50
20 FIELD #1,.....description des zones...
30 FIELD #1,2 AS LOF$

40 GET #1,1:COMPTEUR=CVI(LOF$):IF COMPTEUR<2 THEN COMPTEUR=2
          :
          :
        Saisie

110 PUT #1,COMPTEUR:COMPTEUR=COMPTEUR+1
120 LSET LOF$=MKI$(COMPTEUR):PUT #1,1        ' Sauvegarde compteur
```

- On peut également gérer une BIT-MAP d'occupation fichier qui permet en plus l'allocation dynamique des enregistrements (voir allocation dynamique).

LOF* (N° fichier) *TRS VTOS

LOF indique le nombre de blocs (de 256 octets) utilisés et ne représente donc le nombre d'enregistrements du fichier que si ceux-ci ont une longueur de 256.

***EOF* (N° fichier)**

(5.) est égale à -1 si on référence en lecture un enregistrement au-delà des blocs de 128 octets déjà attribués.

Erreurs classiques

Ces 'erreurs' sont d'autant plus ennuyeuses qu'elles ne sont pas détectées comme erreurs par BASIC.

- Oubli de LSET avant PUT. On ne retrouve donc pas ensuite à la lecture ce que l'on croit avoir écrit dans le fichier.

- (Pour vérifier ce qui a été effectivement écrit dans un enregistrement, le relire par GET# .. en mode DIRECT après l'exécution de PUT#..)

- Utilisation dans une instruction 'INPUT' d'une variable précédemment définie dans un 'FIELD'. Celle-ci se trouve alors désallouée du buffer correspondant.

- Une même variable a été utilisée dans plusieurs FIELD. Elle n'est affectée en réalité qu'au dernier FIELD exécuté.

- Chevauchement (sans le vouloir) des zones. (par des FIELD#.. multiples).

- On ne lit pas les informations suivant le type où elles ont été rangées. (par exemple une information écrite par MKIØ doit être convertie à la lecture par CVI).

- Oubli de GET# avant la modification d'une zone d'un enregistrement. La zone modifiée est bien retrouvée mais toutes les autres contiennent les valeurs de l'enregistrement présent dans le buffer au moment de la mise à jour.

```
10 PRINT "ARAND le 23.10.79":PRINT
20 '
30 '     FICHIERS A ACCES DIRECT (RANDOM)   CREATION,CONSULTATION,MODIFICATION
40 '     ================================
50 '
60 '   LES FICHIERS PERMETTENT DE CONSERVER DES INFORMATIONS
70 '   (LES TABLES N'ETANT QUE DES MEMOIRES DE TRAVAIL)
80 '
110 '
120 '      ----------------------------
130 '    1 ¦ DUPONT ¦JEAN  ¦955-06-46 ¦  <----- ENREGISTREMENT 1
140 '      ----------------------------
150 '    2 ¦        ¦      ¦          ¦
160 '      ----------------------------
170 '    3 ¦        ¦      ¦          ¦
180 '      ----------------------------
190 '         N$      PN$      TL$       <------- ZONES
200 '
210 '       CONSTITUTION D'UN FICHIER
220 '       =========================
230 '
240 REM       1) OUVERTURE DU FICHIER (OPEN)
250 REM       2) DESCRIPTION DES ZONES DES ENREGISTREMENTS   (FIELD # ...)
260 REM       3) ECRITURE       (PUT #....)
270 '
280 OPEN "R",1,"CLIEN"           ' Ouverture du fichier 'CLIEN' sous le no 1
290 FIELD #1,10 AS N$,8 AS PN$,20 AS TL$,4 AS MT$      ' Definition des zones
300 '
320 PRINT:INPUT "      Mode? (CREAT,CONS,MODI,RN,FIN,LIST) ";MODE$
325 '
330 IF MODE$="CREAT" THEN GOSUB 420    ' Création d'un enregistrement
340 IF MODE$="RN"    THEN GOSUB 830    ' Recherche d'un nom
350 IF MODE$="LIST"  THEN GOSUB 960    ' Liste du fichier
360 IF MODE$="CONS"  THEN GOSUB 590    ' Recherche  enregistrement par un no
370 IF MODE$="FIN"   THEN CLOSE #1:STOP
380 IF MODE$="MODI"  THEN GOSUB 700    ' Modification d'un enregistrement
390 GOTO 320
400 '
410 '-----------------------------------------------------------------------
415 '                                                                  CREATION
420 ARANG=LOF(1):GET #1,ARANG                  ' Ajout en fin de fichier=======
430 PRINT:INPUT "Nom?       ";NOM$             ' Entrer le NOM
440 IF NOM$="" GOTO 560                        ' Fin de mode?
450 LSET N$=NOM$                               ' Transfert de NOM$ DANS N$
460 '
470 INPUT "Prenom?    ";PR$
480 LSET PN$=PR$                               ' Transfert de PR$ dans PN$
490 '
500 INPUT "Telephone? ";TEL$:LSET TL$=TEL$
510 INPUT "Matricule? ";MAT:LSET MT$=MKS$(MAT)
520 '
530 PUT #1,ARANG                               ' Ecriture dans le fichier no 1
535 '                                          ' dans enregistrement no ARANG
540 PRINT:PRINT TAB(30) "J'ai range  votre NOM en:";ARANG:PRINT
550 GOTO 420
560 RETURN
580 '-----------------------------------------------------------------------
590      PRINT:INPUT "Quel enregistrement cherchez vous? ";X     ' CONSULTATION
600      IF X=0 GOTO 640                                          ============
610      GET #1,X
620      PRINT:PRINT N$,PN$,TL$,CVS(MT$)       ' Impression des zones
630      GOTO 590
640      RETURN
650 '
```

```
680 '
690 '-----------------------------------------------------------------------
700 PRINT:INPUT "Quel ENREGISTREMENT voulez-vous modifier? ";ARANG:
          IF ARANG=0 GOTO 790
710 '
720 GET #1,ARANG                                                MODIFICATION
730 PRINT                                                '      ============
740 PRINT USING "\            \"; N$;:INPUT X$:IF X$><"" THEN LSET N$=X$
750 PRINT USING "\            \";PN$;:INPUT X$:IF X$><"" THEN LSET PN$=X$
760 PRINT USING "        ######";CVS(MT$);:INPUT X:
                              IF X><0 THEN LSET MT$=MKS$(X)
770 PUT #1,ARANG                              ' Réécriture
780 GOTO 700
790 RETURN
810 '-----------------------------------------------------------------------
820 '                                                       RECHERCHE PAR NOM
830 PRINT:INPUT " NOM cherché? ";X$:IF X$="" GOTO 930   '   ================
835 FF=LOF(1)-1                               ' FF=fin de fichier
836 '
840 FOR I=1 TO FF                             ' Lecture de tout le fichier
850    GET #1,I
860    IF X$=LEFT$(N$,LEN(X$)) THEN I=FF:NEXT I:GOTO 910
870 NEXT I
880 '
890 PRINT:PRINT "    N'existe pas":PRINT:GOTO 830
900 '
910 PRINT:PRINT N$,PN$,TL$,CVS(MT$)           ' Le nom est trouvé
920 GOTO 830
930 RETURN
940 '-----------------------------------------------------------------------
950 '                                                       LISTE DU FICHIER
960 PRINT:PRINT "     LISTE DU FICHIER":PRINT           '   ================
970 '
980  FOR I=1 TO LOF(1)-1
990     GET #1,I
1000    IF ASC(N$)=0 GOTO 1020                ' Enregistrement vide?
1010    PRINT N$,PN$,TL$,CVS(MT$)
1020 NEXT I
1030 '
1040 RETURN
1050 '-----------------------------------------------------------------------
1070 '                  .Avec MICROSOFT 5. voir LOF:
1090 '
1100 '                  .SUR TRS80 NEWDOS:
1110 '                                     CLEAR(3000)
1120 '                                     REMPLACER LOF(1) par LOF(1)+1
1130 '                                     Avant INPUT "xxxx? ";X$  faire X$=""
1140 '                                     OPEN "R","TOTO:1"  (pour unité No 1)
1150 '                                     PENSER à CLORE les FICHIERS
```

CHAPITRE 2-2

FICHIERS SEQUENTIELS

```
.OPEN  PRINT #  INPUT #
 PRINT # USING
.LINE INPUT #
.WRITE #
.CLOSE  EOF
.LOF
.INPUT$
```

Le stockage d'informations s'y fait en fin de fichier au fur et à mesure des arrivées. Une information particulière ne pouvant y être ensuite retrouvée qu'après avoir lu toutes les précédentes (accès séquentiel), leur organisation est dite séquentielle. Les instructions d'écriture et de lecture sont analogues à celles de l'impression et de la lecture au clavier. (PRINT - INPUT).

DUPONT	CR	LF	955-06-46	CR	LF	SALBREUX	CR	LF	296-46-97	CR	LF	GOULMOT	

Dans le cas le plus simple, les ENREGISTREMENTS sont séparés, de façon interne et donc transparente à l'utilisateur, par des caractères 'Carriage Return' (Retour chariot) et 'Line Feed' (saut de ligne) (Codes ASCII 13 et 10).

OPEN

Un fichier séquentiel sur disque avec Microsoft 5. est ouvert soit en sortie (écriture), soit en entrée (lecture) :

```
OPEN "O ou I",#No fichier,"U:Nom fichier" (U=A,B,C,..
                                          représente l'unité
                                          de disque)
```

O : OUTPUT (ECRITURE)
I : INPUT (LECTURE)

```
10 OPEN "O",#1,"A:TOTUS"⟵ MICROSOFT 5.
10 OPEN "O",#1,"TOTUS:1"⟵ sur TRS80 NEWDOS
```

Attention ! avec Microsoft 5., l'ouverture d'un fichier en OUTPUT l'initialise à zéro.

CHAINES DE CARACTERES

PRINT#-INPUT#

Dans le cas le plus simple, l'écriture dans un fichier séquentiel se fait par :

```
PRINT#no fichier,variable
```

Si l'utilisateur programme :

```
 90 INPUT "Nom,Telephone ?  ";NOM$,TPH$
130 PRINT #1,NOM$
140 PRINT #1,TPH$
```

il y a écriture de 2 enregistrements qui seront lus par :

```
INPUT#no fichier,variable
```

```
220 INPUT #1,NOM$
230 INPUT #1,TPH$
250 PRINT NOM$,TPH$
```

Plusieurs enregistrements peuvent être lus simultanément en séparant les variables par des virgules.

```
340 INPUT#1, NOMØ,TPHØ
```

Les lectures doivent, bien sûr, s'effectuer dans l'ordre où les écritures ont été faites.

```
10 '
20 '                    Les FICHIERS SEQUENTIELS
30 '                    ========================
40 '
50 '                                                ' ECRITURE
60 '                                                  --------
70 '
80 OPEN "O",1,"TPH"                        ' Ouverture du fichier TPH sous No 1
90 INPUT "Nom? ";NOM$
100 IF NOM$="" THEN CLOSE #1:GOTO 195
110 INPUT "Tph? ";TPH$
120 '
130    PRINT #1,NOM$          '  <----- Ecriture d'un enregistrement
140    PRINT #1,TPH$          '  <----- Ecriture d'un enregistrement
150 '
160 GOTO 90
170 '-----------------------------------------------------------------------
180 '                                               ' LECTURE
190 '                                                 -------
195 LPRINT "Premier essai":LPRINT
200 OPEN "I",1,"TPH"
210 '
220    INPUT #1,NOM$          '   <---- Lecture d'un enregistrement
230    INPUT #1,TPH$          '   <---- Lecture d'un enregistrement
240 '
250 LPRINT NOM$,TPH$
260 IF EOF(1)=-1 THEN CLOSE #1:LPRINT:GOTO 315      ' Détection FIN de FICHIER
270 GOTO 220
280 '-----------------------------------------------------------------------
300 '                                               ' LECTURE
310 '                                                 -------
315 LPRINT "Deuxième essai:":LPRINT
320 OPEN "I",1,"TPH"
330 '
340    INPUT #1,NOM$,TPH$     '  <----- Lecture de 2 enregistrements
350 '
360 LPRINT NOM$,TPH$
370 IF EOF(1) THEN STOP
380 GOTO 340
390 '-----------------------------------------------------------------------
400 '
410 ' run
420 '
430 ' Nom? LANGEARD
440 ' Tph? 604-91-42
450 ' Nom? ROULET
460 ' Tph? 609-84-59
470 ' Nom?
480 '
490 ' Premier essai:
495 '
500 ' LANGEARD   604-91-42
510 ' ROULET     609-84-59
520 '
530 ' Deuxième essai:
540 '
550 ' LANGEARD   604-91-42
560 ' ROULET     609-84-59
```

L'écriture de plusieurs chaînes dans un même enregistrement se fait en plaçant un point-virgule à la fin de l'instruction PRINT.

Exemple :

```
10 NOM$="DUPONT":PRENOM$="JEAN"
20 PRINT #1,PRENOM$;
25 PRINT #1,NOM$
```

A la lecture nous aurons :

```
100 INPUT #1,X$
110 PRINT X$

RUN
JEANDUPONT
```

Séparateurs d'enregistrements

En plus du 'Carriage return', sont considérés comme séparateurs pour les chaînes de caractères, la virgule et le Line Feed.

La lecture d'un enregistrement par INPUT se passe ainsi :

- Les 'ESPACES', ',' 'LINE FEED' et 'CARRIAGE RETURN' sont ignorés en début de lecture.
- Ensuite, si le premier caractère signifiant est un guillemet (") tous les caractères jusqu'au prochain guillemet sont pris en compte.
- S'il n'y a pas de guillemet, la lecture s'arrête sur un séparateur (, CR LF)

Attention ! : des virgules ',' dans les chaînes seraient considérées comme des séparateurs :

```
 10 X$="45,RUE LA BRUYERE"
 20 PRINT #1,X$
      :
      :
100 INPUT #1,X$      donnerait      X$=45 au lieu de 45,rue la BRUYERE
```

*LINE INPUT#*no fichier, variable chaîne

Permet de lire tout un enregistrement jusqu'au prochain Carriage Return sans tenir compte des séparateurs tels que la ',' ou Line-Feed.

Cependant la lecture s'arrête à 255 caractères si aucun Carriage Return n'a été rencontré.

```
100 LINE INPUT#1,X$
```

Une autre façon de procéder est d'écrire entre guillemets les chaînes comportant des séparateurs :

```
10 PRINT #1,CHR$(34);X$;CHR$(34)
```

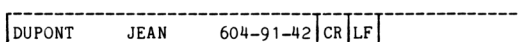
Ainsi à la lecture par INPUT toute la chaîne entre guillemets sera lue.

"DUPONT 45,RUE la BRUYERE"	CR¦LF	

```
100 INPUT #1,X$
110 PRINT X$
```

Donnera DUPONT 45,RUE la BRUYERE

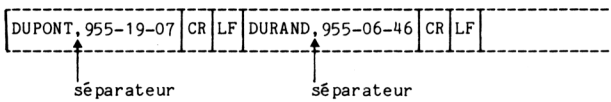
Attention ! PRINT#1,X$,Y$,Z$ n'écrit pas des chaînes séparées par des virgules mais des chaînes séparées par des espaces (comme PRINT sur imprimante)



INPUT #1,X$ donne X$='DUPONT JEAN 604-91-42'

Sous enregistrements

On peut aussi stocker les informations de la façon suivante :




```
10 INPUT "Nom? ";NOM$
20 INPUT "Tel? ";TEL$

30 PRINT #1,NOM$;",";TEL$

40 GOTO 10
```

La lecture pourra plus tard s'effectuer de 2 façons :

```
10 INPUT #1,NOM$,TEL$
20 PRINT NOM$,TEL$

RUN
          DUPONT    955-19-07
```

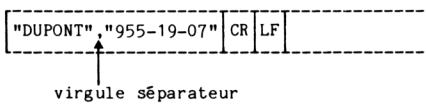
```
10 LINE INPUT #1,X$
20 PRINT X$

   DUPONT,955-19-07
```

WRITE# no fichier,expression1,expression2,..*(**Version** 5.)*

Procède comme PRINT#.. mais sépare les valeurs dans le fichier par des virgules et délimite les chaînes avec des guillemets :

Exemple :

```
10 NOM$="DUPONT":TEL$="955-10-07"
20 WRITE #1,NOM$,TEL$
```



NUMERIQUE

Pour les variables numériques, vient s'ajouter comme séparateur l'espace. La séparation de variables numériques par des ';' ne provoque pas leur concaténation comme pour les chaînes de caractères.

Exemple :

```
10 X=123:Y=345:Z=678
20 PRINT#1,X;Y;Z
```

Ces valeurs dans le fichier sont séparées par des espaces qui seront considérés à la lecture comme des séparateurs pour le numérique :

```
100 INPUT #1,X,Y,Z
110 PRINT X,Y,Z
RUN
123   345  678
```

ECRITURE DE CHAINES ET DE VALEURS NUMERIQUES

Exemple :

```
10 INPUT "Référence ? ";REF$
20 INPUT "Stock     ? ";STOCK
30 PRINT #1,REF$
40 PRINT #1, STOCK

60 GOTO 10
```

On lira ces valeurs par :

```
200 INPUT #1,REF$
210 INPUT #1,STOCK
```

Attention ! ne pas faire :

```
10 X$="CYCLE":X=123
20 PRINT=1, X$;X
```

La lecture de X$ donnerait 'CYCLE 123' comme résultat.

Simulation d'un fichier séquentiel avec le clavier

```
    10 INPUT "Enregistrement? ";X$      ' Lecture d'un pseudo-enregistrement
    20 PRINT X$                         '     au clavier
    30 GOTO 10
run
    Enregistrement? QWERTY

    QWERTY                              (résultat)

    ----------------------------------------------
    Enregistrement? "45,rue la BRUYERE"

    45,rue la BRUYERE                   (résultat)
    ----------------------------------------------
    Enregistrement? QWERTY"JJJJJJ"KKKK

    QWERTY"JJJJJJ"KKKK                  (résultat)
    ----------------------------------------------
    Enregistrement? "45 ,rue la BRUYERE"PARIS

    45,rue la BRUYERE                   (résultat)
    ----------------------------------------------
```

***CLOSE*#no fichier, #no fichier**

En général, avec les fichiers séquentiels, l'écriture sur disque ne se fait pas après chaque PRINT mais seulement lorsque le buffer du fichier est plein. Il importe donc de clore ces fichiers en fin de session.

***EOF*(no fichier)**

Donne -1 si fin de fichier (0 dans le cas contraire) lors d'une lecture de fichier :

Exemple : 100 IF EOF(1)=-1 THEN CLOSE#1:GOTO SUITE

***LOC*(no fichier)**

Indique le nombre de blocs de 128 octets lus ou écrits selon le mode (INPUT ou OUTPUT) (sur 5.)

INPUTS(x,#no fichier) : (Microsoft 5.)

Lit X caractères d'un fichier séquentiel. Avec cette fonction les séparateurs tels que RC,LF.. sont considérés comme les autres caractères.

Exemple : Dump de fichier

```
10 OPEN "I",1,"XX"
20 IF EOF(1)=-1 THEN STOP
30 PRINT HEX$(ASC(INPUT$(1,#1)));" ";
40 GOTO 20
```

PRINT#no fichier USING "format" ; Liste d'expressions

Les règles d'utilisation du PRINT#USING pour les fichiers séquentiels sont les mêmes que pour les PRINT USING sur terminal.

AJOUTS D'ENREGISTREMENTS

Alors que certains fichiers séquentiels acceptent l'ajout d'enregistrements en fin de fichier lors d'une réouverture, d'autres plus rustiques, tels que ceux de Microsoft 5. fonctionnant sous CPM, ne les acceptent pas.

Dans ce cas, l'ajout d'enregistrements se fait en générant un nouveau fichier à l'aide de l'ancien lu en INPUT et des ajouts. Ceci ne simplifie évidemment pas l'exploitation des fichiers séquentiels déjà si peu attrayante.

Les fichiers séquentiels de DTC Microfile, eux, acceptent non seulement les ajouts en fin de fichier mais aussi les insertions et suppressions (et sans décalages grâce à des chaînages internes transparents à l'utilisateur).

IIIème Partie

Méthodes pratiques

CHAPITRE 3-1

QUELQUES METHODES SIMPLES

DATE FICHIER

Il est important, pour travailler sur un fichier de connaître la version utilisée. Une date est, pour ce faire, stockée dans un enregistrement particulier, le n°1 par exemple. Cette date est bien sûr mise à jour à chaque session.

		JO$	MO$	AN$
DATE	1	9	1	80
	2	DUPONT	xxxxxxx	
	3	DURAND	xxxxxxx	

```
10 OPEN "R",1,"CLIENT"
20 FIELD #1,1 AS JO$,1 AS MO$,1 AS AN$            ' Champs pour la date
30 FIELD #1,12 AS NOM$,10 AS PREN$,.............
            .
            .
40 GET #1,1                                       ' Date dans enreg no 1
50 PRINT USING "###";ASC(JO$),ASC(MO$),ASC(AN$)   ' Impression date actuelle

60 PRINT ASC(JO$);:INPUT "Jour ? ";X:IF X>0 AND X<32 THEN
         LSET JO$=CHR$(X) ELSE GOTO 100
70 PRINT ASC(MO$);:INPUT "Mois ? ";X:IF X>0 AND X<13  THEN
         LSET MO$=CHR$(X) ELSE GOTO 100
80 PRINT ASC(AN$);:INPUT "An   ? ";X:IF X>0 AND X<90  THEN LSET AN$=CHR$(X)

100 PUT #1,1

110 DATE$=STR$(ASC(JO$))+STR$(ASC(MO$))+STR$(ASC(AN$))

120 PRINT DATE$                                   ' Impression nouvelle date
```

Une date de création et une date de mise à jour pour chaque enregistrement peuvent également être prévues. Notons que 3 octets suffisent pour coder une date.

LISTE TRIEE D'UN FICHIER A ACCES DIRECT

La plupart du temps, les fichiers ont une taille trop importante pour être amenés en mémoire centrale dans leur totalité, puis triés. On procède donc de la façon suivante :

1- Lire d'abord séquentiellement le fichier en ne conservant en mémoire centrale que les clés à trier dans une table des clés et dans une autre table dite d'index les n° des enregistrements correspondants aux clés lues.

2- Ensuite trier la table des clés, par inversions successives par exemple, en inversant en même temps les n° d'enregistrements correspondants dans la table d'index.

3- Il est alors possible en lisant séquentiellement la table d'index d'obtenir tous les enregistrements du fichier dans l'ordre croissant des clés.

S'il existe un index sur disque en ordre croissant, tenu à jour au fur et à mesure des ajouts, servant aux consultations en temps réel, une liste triée est bien sûr obtenue à l'aide de cet index.

Remarque : Sur cet exemple, nous avons amélioré la méthode de tri 'ripple' en réduisant de un la taille de la table explorée, après chaque passage.

Bubble sort

Cette méthode de tri consiste à comparer d'abord le premier élément à tous les autres (en inversant à chaque fois que l'un de ceux-ci est plus petit que le premier). Le plus petit de la table est ainsi amené en première position. On procède de la même façon en comparant le second élément de la table à tous les éléments restants, etc.

Ainsi, après N explorations de la table (N=nombre éléments), la table est en ordre :

```
10 FOR I=1 TO N-1                  ' BUBBLE SORT
20   FOR J=I+1 TO N                ' Comparaison du Ieme à tous les autres
30     IF A(J)<A(I) THEN SWAP A(I),A(J)
40   NEXT J
50 NEXT I
```

Pour les valeurs aléatoires, les nombres de comparaisons et d'inversions sont les mêmes qu'avec 'ripple amélioré'. Cependant, le temps d'exécution est sensiblement plus faible puisqu'il n'y a pas à positionner le témoin d'inversion.

Des 3 méthodes de tri les plus simples, 'ripple', 'bubble' et 'shell', c'est cette dernière qui est la plus rapide.

Shell-Metzner

Signalons que la méthode de Shell-Metzner est 2 fois plus rapide que celle de Shell (3 fois moins de comparaisons) :

```
100 TAILLE=100:PAS=TAILLE          ' SHELL-METZNER
105 '
110 PAS=INT(PAS/2):IF PAS<1 THEN STOP
120 J=1:K=TAIILE-PAS
125 '
130 I=J
140 '
150 L=I+PAS
160 IF A(I)<A(L) THEN GOTO 200
170 SWAP A(L),A(I)
180 I=I-PAS:IF I<1 GOTO 200 ELSE GOTO 150
190 '
200 J=J+1:IF J>K GOTO 110 ELSE GOTO 130
```

Il pourrait sembler intéressant, pour trier des chaînes de caractères, d'utiliser un index (pointant vers la table des chaînes) où les inversions seraient effectuées plutôt que dans la table des chaînes. En fait, le temps de tri ne varie pratiquement pas.

Cette méthode peut cependant présenter l'avantage de ne pas modifier la table des clés à trier.

Comparaisons de tris : (BASIC interprété sur 8080)

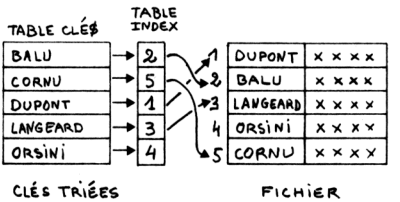
Temps de tri (en secondes) pour des nombres aléatoires entre 0 et 1 :

	BUBBLE	SHELL	SHELL-METZNER	INSERTION DICHOTOMIQUE(OI no 8)
50	15	15	9	15
100	55	45	21	45
200	215	110	50	145
300	470	165	80	300
400			125	470
500			145	760

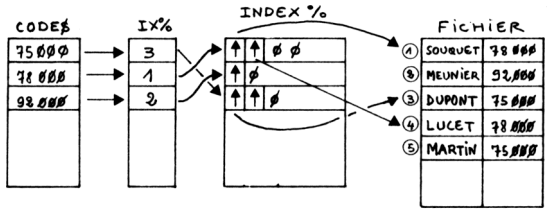
Pour 500 noms de 12 caractères, le temps de tri par Shell-Metzner passe de 145 à 165 secondes.

Précisons que l'ajout dans une liste déjà en ordre de nouveaux éléments, peut être faite en insérant ceux-ci de façon à maintenir la liste en ordre. Si, toujours pour ajouter des éléments dans une liste en ordre, on procède à l'aide d'un tri classique, un 'ripple' (dans les 2 sens) pourra se révéler plus efficace qu'un Shell-Metzner.

```
10 '          LISTE  TRIEE D'UN FICHIER à ACCES DIRECT
20 '          ========================================
30 '
40 '    Au cours d'une première lecture séquentielle  du fichier,
50 '    seules les clés à trier(les noms par ex)
60 '    et les numéros des enregistrements ou elles ont été lues sont gardés
70 '    en mémoire centrale dans des tables(table des clés et table d'index).
80 '    Après avoir trié celles-ci ,on lit à nouveau le fichier,mais
90 '    par l'intermédiaire de l'index que l'on explore séquentiellement.
100 '   On obtient ainsi une liste du fichier dans l'ordre des clés triées.
110 '
120 OPEN "R",1,"CLIEN"                    ' Ouverture du fichier CLIEN
130 FIELD #1,10 AS N$,8 AS PN$,20 AS TL$  ' Définition des ZONES
140 '-------------------------------------------------------------------------
150 '                              ' CONSTITUTION DES TABLES DES CLES ET D'INDEX
160 NB=LOF(1)-1:J=1                       ' NB:nombre d'enregistrements
170 DIM CLE$(NB),INDEX(NB)                ' J :nombre de clés
180 '
190 FOR I=1 TO NB
200    GET #1,I                           ' Lecture de l'enregistrement no I
210    IF ASC(N$)=0 GOTO 240              ' Enregistrement vide?
220    CLE$(J)=N$:INDEX(J)=I:J=J+1        ' Ajout dans la table des clés
230 '  -------------------------          ' et dans la table d'index
240 NEXT I
250 '
260 '-------------------------------------------------------------------------
270 K=J-1                                 ' TRI DES TABLES DES CLES ET D'INDEX
280 INVERSION=0                           '       ('Ripple amélioré')
290 '
300 FOR I=1 TO K-1
310    IF CLE$(I+1)<CLE$(I) THEN
          SWAP CLE$(I+1),CLE$(I):SWAP INDEX(I+1),INDEX(I):INVERSION=1
320 NEXT I
330 '
340 IF INVERSION=1 THEN K=K-1:GOTO 280
350 '
360 '-------------------------------------------------------------------------
370 LPRINT "RESULTAT"                                                  ' EDITION
380 LPRINT "========":LPRINT ""
390 '
400 FOR I=1 TO J-1
410    GET #1,INDEX(I)
420 '  ----------------
430    LPRINT N$,PN$,TL$,INDEX(I)
440 NEXT I
450 '
460 ' run
470 '
480 '  RESULTAT
490 '  ========
500 ' BALU        Thierry    739-33-90     2
510 ' CORNU       Elisabeth  609-00-99     5
520 ' DUPONT      Jean       777-00-66     1
530 ' LANGEARD    Nicole     604-91-42     3
540 ' ORSINI      Bruno      955-11-33     4
```



```
10 '          REGROUPEMENT PAR CODE
20 '
30 '     .Ce programme permet de regrouper pour CHAQUE CODE POSTAL
40 '      toutes les personnes ayant ce code postal.Le regroupement se fait par
50 '      la constitution d'un index (INDEX%) en mémoire centrale contenant
60 '      dans chaque 'ligne' des pointeurs vers tous les enregistrements
70 '      ayant le même code postal.
80 '     .2 explorations de fichier seulement sont nécessaires:1 pour la cons-
90 '      titution de l'index et 1 pour l'édition de la liste
100 '     Afin d'obtenir ces regroupements dans l'ordre des codes ,
110 '     on constitue ,au fur et à mesure des lectures dans le fichier,
120 '     une table des codes et une table d'index IX% pointant vers
130 '     INDEX% et contenant au départ (1,2,3,...) puis on tri la table
140 '     des codes(ainsi que IX%) et enfin on lit INDEX% via IX%)
150 '    .INDEX%  crée sur disque(1 enregistrement de 128 octets
160 '     contiendrait 64 pointeurs), permettrait plus tard, grâce à la
170 '     mémorisation des regroupements, de faire des recherches en
180 '     'temps réel' plus rapides(recherche du type :
190 '     'donnez-moi toutes les personnes ayant tel code postal')
200 '
210 DIM INDEX%(50,20),CODE$(50),IX%(50)                ' 50 codes maxi
220 OPEN "R",1,"NOM":FIELD #1,12 AS NOM$,5 AS CPOST$
330 '-----------------------------------------------------------------------
340 FOR NE=1 TO LOF(1)                                 ' CONSTITUTION DE L'INDEX
350    GET #1,NE:IF ASC(NOM$)=0 GOTO 480
360 '
370    FOR K=1 TO 50                                   ' Recherche code
380      IF CPOST$=CODE$(K) THEN GOTO 430
390      IF CODE$(K)="" THEN CODE$(K)=CPOST$:IX%(K)=K:NB=NB+1:GOTO 430
400    NEXT K
410    PRINT "Augmentez la taille de CODE$-IX%-INDEX%":STOP
420 '
430    FOR I=1 TO 20                                   ' Ajout pointeur
440      IF INDEX%(K,I)=0 THEN INDEX%(K ,I)=NE:GOTO 480
450    NEXT I
460    PRINT "Augmentez la taille de INDEX%":STOP
470 '
480 NEXT NE
490 '-----------------------------------------------------------------------
500   FOR I=1 TO NB-1                                  ' TRI DES CODES
510     FOR J=I+1 TO NB                                '   ('bubble')
520       IF CODE$(J)<CODE$(I) THEN SWAP CODE$(I),CODE$(J):
                 SWAP IX%(I),IX%(J)
530     NEXT J
540   NEXT I
550 '-----------------------------------------------------------------------
560 LPRINT :LPRINT "Code";TAB(20)"Nom":LPRINT
570 FOR CODE =1 TO 50                                  ' LISTE PAR CODE
580    LPRINT CODE$(CODE);
590 '
600    FOR I=1 TO 20
610      NE=INDEX%(IX%(CODE),I):IF NE=0 GOTO 650
620      GET #1,NE:LPRINT TAB(20) NOM$
630    NEXT I
640 '
650    LPRINT
660 NEXT CODE
670 '
680 ' run
690 '
700 ' Code       Nom
710 '
720 ' 75000      DUPONT
730 '            MARTIN
740 '
750 ' 78000      SOUQUET
760 '            LUCET
```

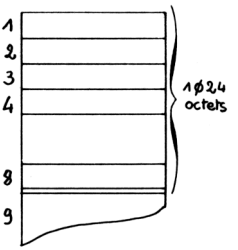


ALLOCATION DE L'ESPACE DISQUE POUR LES FICHIERS RANDOM

Afin d'éviter d'avoir à réorganiser périodiquement l'espace disque lorsque des fichiers sont supprimés, c'est généralement par blocs de 1024 octets (non contigus) que se fait l'allocation de l'espace disque.

Un index pour chaque fichier (transparent à l'utilisateur) permet au système de retrouver les différents blocs de 1024 octets alloués à chacun d'eux.

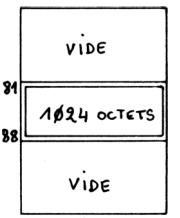
Par exemple, si l'enregistrement n°1 (128 octets) est créé dans un fichier vide, 1024 octets sont réservés pour les enregistrements allant de 1 à 8. Lorsque plus tard, l'enregistrement n°9 sera créé, 1024 octets seront alloués pour les enregistrements 9 à 16.



ALLOCATION DE L'ESPACE DISQUE

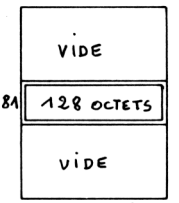
Avec Microsoft 5. l'écriture au n°81 dans un fichier vide, ne provoque une réservation que pour les enregistrements allant de 81 à 88. (pour des enregistrements de 128).

Sur TRS80 NEW-DOS, l'écriture au n°81 dans un fichier vide provoque en plus une réservation pour les 80 premiers enregistrements.



ALLOCATION DISQUE AVEC MICROSOFT 5.

Un système comme DTC MICROFILE ne fait l'allocation que par 128 octets. Ce qui autorise l'écriture dans un fichier de façon aléatoire sans jamais perdre de place disque.
(Voir fichier virtuel).



ALLOCATION DISQUE SUR DTC MICROFILE

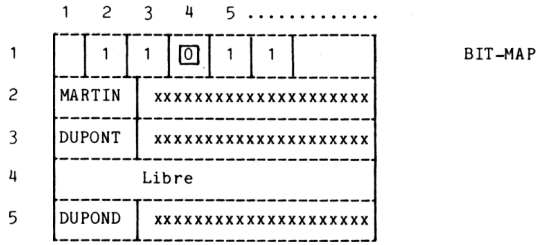
ALLOCATION DYNAMIQUE

L'ajout d'enregistrements dans un fichier RANDOM peut se faire en fin de fichier à l'aide de LOF. Mais lorsque des enregistrements sont supprimés à l'intérieur d'un fichier, pourquoi ne pas les récupérer dynamiquement plutôt que de réorganiser périodiquement ce fichier ?

Deux méthodes existent : le chaînage des enregistrements vides et la gestion d'une 'Bit-Map' (carte de bits) indiquant pour chaque enregistrement dans le fichier s'il est occupé ou non. C'est cette dernière que nous allons étudier.

Pour cela, utilisons l'enregistrement n°1 (256 octets par exemple) comme pseudo Bit-Map. Si nous avons moins de 254 enregistrements à gérer, réservons un octet pour représenter l'occupation de chaque enregistrement.

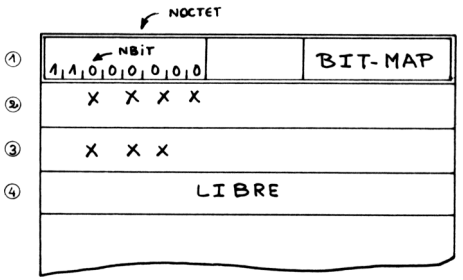
Au départ, tous les octets de la bit-map sont à zéro. A chaque création dans le fichier, nous positionnons à 1 l'octet correspondant à l'enregistrement utilisé.




```
 10 TENREG=254:DIM BMAP$(TENREG)
 20 FOR I=1 TO TENREG:FIELD#1,(I-1)*1 AS D$,1 AS BMAP$(I):NEXT I
 30 FIELD #1,12 AS N$,........

100 GOSUB 1000              ' Appel pour avoir un enregistrement libre

110 INPUT "Nom ? ";NOM$
120 LSET N$=NOM$
130 PUT #1,ARANG            ' ARANG --> adresse de rangement calculee

140 GOSUB 2000              ' Appel MAJ BIT-MAP
150 STOP
990 '-----------------------------------------------------------------------
1000 GET #1,1                                  ' Appel de la BIT-MAP
1005 '
1010 FOR W=2 TO TENREG                         ' Recherche dans la  BIT-MAP
1020   IF ASC(BMAP$(W))=0 THEN ARANG=W:GOTO 1500
1030 NEXT W
1035 '
1040 PRINT "Il n'y a plus d'enregistrement disponible":STOP
1500 RETURN

1990 '-----------------------------------------------------------------------
2000 GET #1,1:LSET BMAP$(ARANG)=CHR$(1):PUT #1,1:RETURN  'MAJ BIT-MAP(ajout)
2010 '-----------------------------------------------------------------------

3000 GET #1,1:LSET BMAP$(ARANG)=CHR$(0):PUT #1,1:RETURN  'MAJ BIT-MAP(sup)
```

Si nous devons gérer un nombre plus important d'enregistrements, nous représentons 8 positions par octet. Nous disposons ainsi par exemple, pour un enregistrement de 100 octets, de 800 positions.

```
10 '        ALLOCATION DYNAMIQUE   (gestion d'une  BIT-MAP)
20 '        ==============================================
30 '
40 DIM BMAP$(50)
50 OPEN "R",1,"JBG":FIELD #1,12 AS N$
60 FOR I=1 TO 50 :FIELD #1,(I-1) AS D$,1 AS BMAP$(I):NEXT I
70 '
80 INPUT "Nom? ";NOM$
90 GOSUB 140:GET #1,ARANG                ' Appel recherche enregistrement
100 LSET N$=NOM$:PUT #1,ARANG
110 GOTO 80
120 '-----------------------------------------------------------------------
130 '                                          RECHERCHE ENREGISTREMENT LIBRE
140 GET #1,1                  ' Appel BIT-MAP
150 FOR I=1 TO 50
160 '                       NOCTET=No octet:NBIT=No bit à l'intérieur de l'octet
170     X%=ASC(BMAP$(I))
180     IF X%=255 GOTO 250               ' Tous les bits de cet octet sont à 1
190 '
200     FOR NBIT=1 TO 8                  ' Recherche de 1 bit a 0 dans l'octet
210       MASQUE%=2^(NBIT-1)
220       IF (X% AND MASQUE%)=0 THEN NOCTET=I:I=50:NEXT I:GOTO 290
230     NEXT NBIT
240 '
250 NEXT I
260 '
270 PRINT "Il n'y a plus d'enregistrement disponible":STOP
280 '
290 LSET BMAP$(NOCTET)=CHR$(X%+MASQUE%) ' On a trouvé un enregistrement libre
300 ARANG=((NOCTET-1)*8)+NBIT+1          ' Calcul adresse enregistrement libre
310 PRINT ARANG
320 PUT #1,1                             ' Rangement BIT-MAP
330 RETURN
340 '-----------------------------------------------------------------------
350 '                                          SUPPRESSION ENREGISTREMENT
360 GET #1,1                ' Appel BIT-MAP
370 NBIT=(ARANG-1) MOD 8:NOCTET=INT((ARANG-1)/8)+1
380 LSET BMAP$(NOCTET)=CHR$(ASC(BMAP$(NOCTET))-(2^(NBIT-1)))
390 PUT #1,1
400 RETURN
410 '-----------------------------------------------------------------------
420 ' Pour plus de sécurité,nous pourrions avant d'utiliser un enregistrement
430 ' supposé libre,nous assurer qu'il l'est effectivement,en testant son
440 ' contenu.
```



FICHIER A ACCES DIRECT VIRTUEL (DTC MICROFILE)

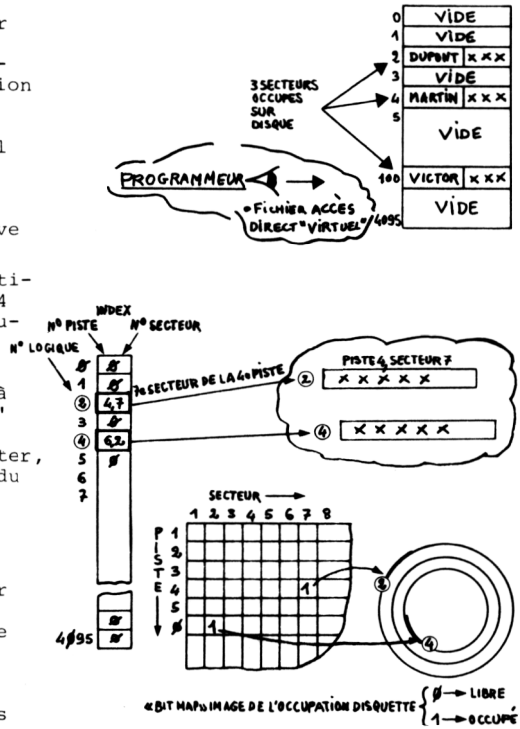
Ce fichier se présente comme un fichier à accès direct pour l'utilisateur ; celui-ci dispose à la création d'un fichier de 4096 enregistrements logiques vides mais il n'y a d'occupation physique sur disque qu'au fur et à mesure de l'écriture effective d'enregistrements.

Par exemple, si l'utilisateur écrit en 2, 4 et 100, 3 secteurs seulement seront occupés sur disque.

Ce type de fichier à accès direct 'virtuel' a parmi ses nombreux avantages celui d'éviter, en cas d'utilisation du hash-code, une perte d'espace disque.

Organisation physique

Un index permet pour chaque fichier d'établir la correspondance entre les n° logiques (seuls connus du programmeur) et les adresses physiques des enregistrements sur disque.

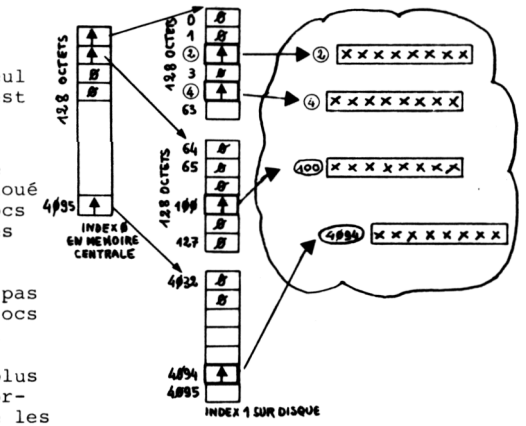


ACCÈS INDEXÉ NUMÉRIQUE (PRINCIPE)

En réalité l'index n'est pas continu, il est à 2 niveaux et seul l'index de niveau 0 est résident en mémoire centrale.

L'index 1 se trouve sur disque et est alloué dynamiquement par blocs de 128 octets capables de recevoir chacun 64 pointeurs.

Notons qu'il n'y a pas de décalages entre blocs comme pour un index à 2 niveaux classique.
Sa gestion est donc plus simple (et plus performante) mais il limite les clés numériques à 4096.

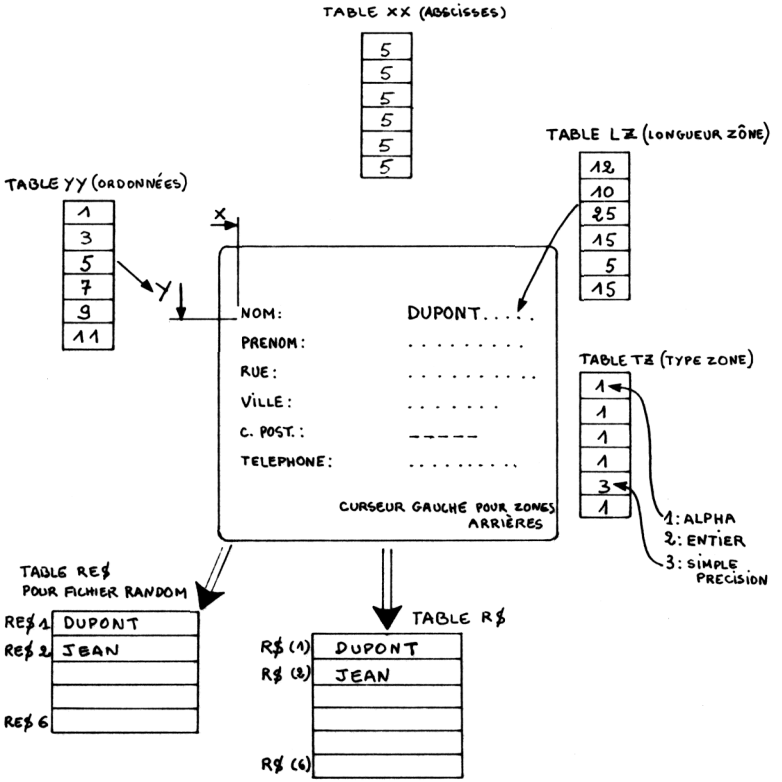


ACCÈS INDEXÉ NUMÉRIQUE (SCHÉMA RÉEL)

SAISIE ECRAN TRS80

Il est intéressant, lors d'une saisie de données, d'en controler la validité au fur et à mesure de leur introduction, et si possible caractère par caractère.

Un système tel que TRS80, muni d'une fonction de saisie d'un caractère INKEY$ le permet. (La fonction INPUT ne le permet pas).



Le positionnement direct du curseur sur les zones à saisir se fait grâce à la fonction :

PRINT @(Yx64)+X où X et Y représentent respectivement l'abscisse et l'ordonnée du curseur.

Le clavier est lu en permanence par INKEY$. On sait si un caractère a été frappé par le test suivant :

```
370  X$=INKEY$
380  IF X$="" GOTO 370   'Si X$="" on n'a rien frappé
      .
430  PRINT X$;           'Impression si caractère valide
```

L'acquisition d'une ZONE se fait par concaténations successives des caractères acquis :

```
440 Y$=Y$+X$
```

La suppression du dernier caractère acquis, lorsque l'opérateur a frappé la touche 'curseur gauche' (code ASCII 8) se fait par :

```
410 IF ASC(X$)=8 THEN .... Y$=LEFT$(X$,LEN(Y$)-1)
```

Le curseur n'apparaissant pas lorsque la FONCTION INKEY$ est utilisée, il nous faut en gérer un par programme. Ce qui nous conduit à afficher le caractère acquis X$ en adressage absolu :

```
430 PRINT@XY+LEN(Y$),X$;
```

Sans cette gestion du curseur, l'instruction : 430 PRINT X$; suffirait. (une gestion de curseur par ligne seulement pourrait être faite à l'aide de SET, RESET).

A chaque fois que la saisie d'une zone se termine par la frappe d'un retour chariot (code ASCII 13), nous la réaffichons.

```
540 ON TZ(I) GOSUB 550,560,560
550 PRINT R$(I);STRING$(LZ(I)-LEN(R$(I)),"."):RETURN
560 PRINT R$(I);STRING$(LZ(I)-LEN(R$(I)),"-"):RETURN
```

Dès qu'une zone saisie atteint la longueur maximum prévue, il y a saut automatique à la zone suivante :

```
450 IF LEN(Y$) >=LZ(I) GOTO 480
```

Lorsque nous appuyons sur la touche 'curseur gauche', et tant que la chaîne Y$ n'est pas vide, il y a suppression des derniers caractères dans celle-ci. Mais dès que Y$ devient vide, la frappe de 'curseur gauche' provoque un retour sur la zone précédent celle où nous sommes positionnés. (instruction 400).

Pour une saisie directe dans un fichier Random, la table RE$, où les résultats sont rangés, doit être déclarée dans l'instruction FIELD ....

3 types de zones sont prévus :
1 ALPHA
2 INTEGER
3 SIMPLE PRECISION

Les fonctions de contrôle ALPHA et NUMERIQUES sont appelées par :

```
420 ON TZ(I) GOSUB 580, 590, 590
```

Des sous-programmes de contrôle "personnalisés" pour certaines zones peuvent être inclus. Rien n'empêche de prévoir une touche particulière pour annuler une zone ou de revenir sur une zone arrière autrement que par 'curseur gauche'.

Si ce sous-programme doit aussi servir à modifier des zones d'enregistrements de fichier Random, l'ancienne valeur de chaque zone est affichée, avant la saisie :

```
295 ON TZ(I) GOSUB 640, 645, 650
```

```
10 PRINT "ECR 30.10.79":PRINT
20 '
30 '            SAISIE ECRAN TRS80 CARACTERE PAR CARACTERE
40 '
50 '                          CURSEUR CARACTERE par CARACTERE
60 '                          ==============================
70 CLEAR(3000):NZ=6
90 '                       Résultat dans tables R$ et RE$
110 '
120 DATA 1,3,5,7,9,11:     FOR I=1 TO NZ:READ YY(I):NEXT I   ' Ordonnées
130 DATA 5,5,5,5,5,5:      FOR I=1 TO NZ:READ XX(I):NEXT I   ' Abscisses
150 DATA 1,1,1,1,3,1 :     FOR I=1 TO NZ:READ TZ(I):NEXT I   ' Types zones
160 DATA 20,10,25,15,5,15:FOR I=1 TO NZ:READ(LZ(I):NEXT I   ' Longueurs zones
165 DATA "NOM:","PRENOM:","RUE:","VILLE:","CPOST:","TELEPH:":
                     FOR I=1 TO NZ:READ NZ$(I):NEXT I
190 GOSUB 250:STOP
200 '-----------------------------------------------------------------------
210 '        SAISIE ECRAN                Pour fichier RANDOM :
220 '                                    déclarer table RE$ dans FIELD # .....
230 '                   295 PRINT @XY+10,"";:ON TZ(I) GOSUB 640,645 ,650
240 '
250 FOR I=1 TO NZ:R$(I)="":NEXT I
260 CLS
270    FOR I=1 TO NZ                                          ' Affichage masque
280      XY=YY(I)*64+XX(I):PRINT @XY,NZ$(I);
300      PRINT @XY+34,"";:ON TZ(I) GOSUB 550,560,560,560
310    NEXT I
320 '
330 PRINT @924," Curseur gauche pour ZONES ARRIERES"
340 I=1:Y$=""
350 XY=YY(I)*64+XX(I)+34:IF TZ(I)=1 THEN AE$="." ELSE AE$="-"
355 '
360 PRINT @XY+LEN(Y$),CHR$(136);                              ' Affichage curseur
370 X$=INKEY$                                                 ' Lecture du clavier
380 IF X$="" GOTO 370                                         ' Test du clavier
390 IF ASC(X$)=13 THEN IF Y$="" THEN  490 ELSE GOTO 480       ' Retour Chariot?
400 IF ASC(X$)=8 THEN IF LEN(Y$)=0 THEN
          IF I>1 THEN GOSUB 540:I=I-1:GOTO 350 ELSE GOTO 370
410 IF ASC(X$)=8 THEN PRINT @XY+LEN(Y$),AE$;:Y$=LEFT$(Y$,LEN(Y$)-1):GOTO 360
420 ON TZ(I) GOSUB 580,590 ,590,590:ON Q GOTO 430,370         ' Appel contrôle
430 PRINT @XY+LEN(Y$),X$;                                     ' Imp caract acquis
440 Y$=Y$+X$                                                  ' Ajout du caractère
450 IF LEN(Y$)=>LZ(I) GOTO 480                                ' Fin de zone ?
460 GOTO 360
470 '
480 R$(I)=Y$:ON TZ(I) GOSUB 610,615,620:Y$=""                 ' Rangement zone
490 IF I>=NZ THEN RETURN                                      ' Fin de saisie?
510 GOSUB 540:I=I+1:GOTO 350                                  ' Réaffichage zone
530 '
540 PRINT @XY,"";:ON TZ(I) GOSUB 550,560,560,560:RETURN
550 PRINT R$(I);STRING$(LZ(I)-LEN(R$(I)),"."):RETURN          ' Affichage zone
560 PRINT R$(I);STRING$(LZ(I)-LEN(R$(I)),"-"):RETURN
570 '
580 IF ASC(X$)<32 THEN Q=2:RETURN ELSE Q=1:RETURN             ' Contrôle ALPHA
590 IF ASC(X$)<48 OR ASC(X$)>57 THEN Q=2:RETURN
                                     ELSE Q=1:RETURN          ' Contrôle NUMERIQUE
610 LSET RE$(I)=Y$:RETURN                                     ' Rangement fichier
615 LSET RE$(I)=MKI$(VAL(Y$)):RETURN
620 LSET RE$(I)=MKS$(VAL(Y$)):RETURN
630 '
640 PRINT RE$(I);:RETURN                                      ' Affichage ancienne
645 PRINT CVI(RE$(I));:RETURN                                 ' zone
650 PRINT CVS(RE$(I));:RETURN
```

Saisie sur écran indépendant

Lorsqu'un écran est indépendant d'un système (contrairement au TRS80 par exemple), c'est par l'envoi de "caractères de contrôle" que se fait sa gestion :

par exemple l'effacement de l'écran se programme :

```
10 PRINT CHRØ(X)   X ayant une valeur propre à chaque
                   type d'écran.
```

Le positionnement direct du curseur dans l'écran se fait en envoyant un caractère de contrôle spécifique immédiatement suivi des coordonnées X et Y

```
PRINT CHRØ(XY);CHRØ(X);CHRØ(Y)
```

valeur spécifique au type d'écran - abscisse - ordonnée

Il est parfois nécessaire d'ajouter à X et Y une constante (32 sur l'exemple).

Quelques caractères de contrôle sont standard :

Ainsi PRINT CHRØ(7) active la sonnerie de l'écran.

Parmi les caractères envoyés par le clavier, le curseur gauche a pour code 8, le retour chariot le code 13.

En BASIC Microsoft 5., c'est la fonction INPUTØ(1) qui permet d'acquérir les données caractère par caractère :

```
XØ=INPUTØ(1)
```

Les caractères de contrôle de l'écran du LX500 sont :

Effacement:	PRINT CHR$(133)
Video inverse:	PRINT CHR$(29)
Fin video :	PRINT CHR$(28)
Adressage XY:	PRINT CHR$(27)+CHR$(61)+CHR$(32+Y)+CHR$(32+X)

Pour cet écran, les coordonnées de positionnement du curseur doivent être envoyées dans l'ordre Y,X au lieu de X,Y.

Si un programme de saisie doit fonctionner avec différents types d'écrans, on demandera à l'opérateur en début de session sur quel type d'écran il travaille et on affectera alors aux différents caractères de contrôle les valeurs adaptées au type d'écran utilisé.

```
INPUT "Type ecran? ";ECRAN$
IF ECRAN$="TYPE1" THEN XY$=CHR$(XX):DEF FNXY$=.....
IF ECRAN$="TYPE2" THEN XY$=CHR$(YY):DEF FNXY$=.....
```

```
10 '                    SAISIE  ECRAN (ET MODIFICATION)    LX500
20 '                    ==============================
30 '                                          RESULTAT DANS R$ ET RESUL$
40 OPEN "R",1,"NOM"
50 '
60 EFFAC$=CHR$(133):CG$=CHR$(8):XY$=CHR$(27)+CHR$(61):VIDEO$=CHR$(29):
                            FVIDEO$=CHR$(28):BX=32:RC$=CHR$(13):SON$=CHR$(7)
70 '
80 NZ=5:DIM RESUL$(NZ),R$(NZ),NZ$(NZ)                    ' NZ:nombre de zones
90 DATA "NOM:","PRENOM:","RUE:","VILLE:","CPOST:":
   FOR I=1 TO NZ:READ NZ$(I):NEXT I
100 DATA 2,2,2,2,2:   FOR I=1 TO NZ:READ XX(I):NEXT I   ' Abscisses
110 DATA 3,5,7,9,11:  FOR I=1 TO NZ:READ YY(I):NEXT I   ' Ordonnees
120 DATA 1,1,1,1,3:   FOR I=1 TO NZ:READ TZ(I):NEXT I   ' Types des zones
130 DATA 12,5,25,10,5:FOR I=1 TO NZ:READ LZ(I):NEXT I   ' Longueurs des zones
140 '
150 FIELD #1,12 AS RESUL$(1),12 AS RESUL$(2),25 AS RESUL$(3),15 AS RESUL$(4),
                      4 AS RESUL$(5)
160 DEF FNXY$(X,Y)=XY$+CHR$(BX+Y)+CHR$(BX+X)     ' Adressage curseur
170 GET #1,100: GOSUB 190:PUT #1,100:STOP        ' Appel sous programme saisie
180 '-----------------------------------------------------------------------
190 PRINT EFFAC$;:FOR I=1 TO NZ:R$(I)="":NEXT I
200 '
210   FOR I=1 TO NZ
220     PRINT FNXY$(XX(I),YY(I));VIDEO$;:PRINT USING "\       \";NZ$(I);
230     PRINT FNXY$(XX(I)+15,YY(I));FVIDEO$;
240     ON TZ(I) GOSUB 560,570,580,450           ' Affichage ancienne zone
250     PRINT FNXY$(XX(I)+35,YY(I));
260     ON TZ(I) GOSUB  450,460,460,460          ' Affichage masque
270   NEXT I
280 '
290 I=1:Y$=""                                    ' Y$=Chaîne courante
300 PRINT FNXY$(XX(I)+35,YY(I));:IF TZ(I)=1 THEN AE$="." ELSE AE$="-"
310 '
320 X$=INPUT$(1)                                 ' Lecture clavier
330 IF X$=CG$ AND LEN(Y$)=0 THEN PRINT SON$;:
              IF I>1 THEN GOSUB 440:I=I-1:Y$="":GOTO 300 ELSE GOTO 320
340 IF X$=CG$ THEN
            Y$=LEFT$(Y$,LEN(Y$)-1):PRINT CG$;AE$;CG$;:GOTO 320  'Sup 1 caract
350 IF X$=RC$ THEN IF Y$="" THEN GOTO 410  ELSE GOTO 400
360 ON TZ(I) GOSUB 480,490,490,490:ON Q GOTO 370,320,320,320
370 Y$=Y$+X$:PRINT X$;
380 IF LEN(Y$)< LZ(I) THEN GOTO 320  ELSE PRINT SON$;     ' Fin de zone?
390 '
400 R$(I)=Y$:ON TZ(I) GOSUB 510,520,530,540:Y$=""        ' Rangement zone
410 GOSUB 440:IF I>=NZ THEN RETURN                       ' Fin saisie?
420 I=I+1:GOTO 300
430 '
440 PRINT FNXY$(XX(I)+35,YY(I));:ON TZ(I) GOSUB 450,460,460,460:RETURN
450 PRINT R$(I);STRING$(LZ(I)-LEN(R$(I)),".")):RETURN
460 PRINT R$(I);STRING$(LZ(I)-LEN(R$(I)),"-"):RETURN
470 '
480 IF ASC(X$)<32 THEN PRINT SON$;:Q=2:RETURN ELSE :Q=1:RETURN     'Contrôle
490 IF ASC(X$)<48 OR ASC(X$)>57 THEN PRINT SON$;:Q=2:RETURN ELSE Q=1:RETURN
500 '
510 LSET RESUL$(I)=Y$:RETURN                     ' Saisie dans fichier RANDOM
520 LSET RESUL$(I)=MKI$(VAL(Y$)):RETURN
530 LSET RESUL$(I)=MKS$(VAL(Y$)):RETURN
540 LSET RESUL$(I)=CHR$(VAL(Y$)):RETURN
550 '
560 PRINT RESUL$(I);:RETURN                      ' Affichage ancienne valeur
570 PRINT CVI(RESUL$(I));:RETURN
580 PRINT CVS(RESUL$(I));:RETURN
```

CHAPITRE 3-2

L'ACCES INDEXE

La plupart des P.S.I. de gestion actuels proposent des fichiers avec un accès par un n° d'enregistrement. Or, en gestion, c'est par un nom ou par un code que l'utilisateur désire accéder aux informations.

Il serait possible, bien sûr, de rechercher une information dans un fichier par un nom ou par un code, en lisant séquentiellement tous les enregistrements, mais compte tenu du temps d'accès, de l'ordre de 100 ms, pour chaque enregistrement, l'information cherchée ne serait retrouvée qu'après plusieurs secondes pour un fichier de taille moyenne.

Comment établir la correspondance entre les clés d'accès de l'utilisateur et les n° d'enregistrements où sont rangées les informations afin de limiter les accès disque et de réduire ainsi les temps d'accès ?

Deux principes sont généralement utilisés :

- Soit le Hash-Code qui établit la correspondance par un calcul d'adresse de rangement effectué sur la clé ;

- Soit l'accès indexé qui établit la correspondance à l'aide d'une table.

Si la première méthode est plus simple dans son principe, elle est aussi plus coûteuse en place disque, du moins avec les fichiers classiques.

C'est donc vers l'accès indexé que nous orienterons nos recherches.

- Accès indexé simple : table en mémoire centrale constituée en début de session.

- Accès indexé avec sauvegarde de la table d'index sur disque.

- Accès indexé avec index continu sur disque en ordre croissant.

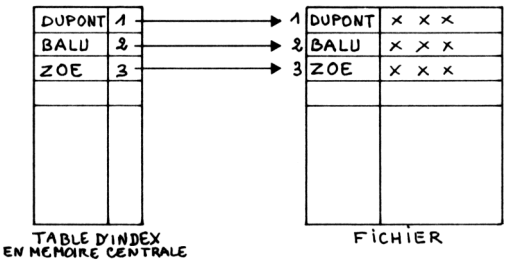
- Accès indexé avec index à deux niveaux.

- Exemple d'index TRS 80.

De nombreuses méthodes existent, plus ou moins simples à programmer. Leur choix se fait en fonction des exigences de l'application et des particularités des fichiers utilisés.

ACCES INDEXE AVEC TABLE DE CORRESPONDANCE EN MEMOIRE CENTRALE

Dans la version la plus simple de l'accès indexé, la table de correspondance est résidente en mémoire centrale et se constitue en début de session par la lecture séquentielle de tout le fichier :



Lorsque cette table n'est pas triée, sa consultation est nécessairement séquentielle mais relativement rapide si les clés ne sont pas trop nombreuses. Il suffit ensuite d'un accès disque pour retrouver l'enregistrement associé à la clé.

Mais plusieurs dizaines de secondes sont nécessaires à la constitution de la table si le fichier a une taille importante. Aussi cette méthode n'est guère exploitable dès que le nombre d'enregistrements physiques du fichier devient important (au-delà d'une centaine). Cependant, elle a l'avantage d'être simple à programmer et aussi celui de s'adapter aisément sur une application existante par l'appel de 2 sous-programmes : l'un de création de l'index, l'autre de recherche de la clé.

Si la table d'index est en ordre croissant, la recherche s'y fait par dichotomie et devient ainsi plus rapide.

Seules les premières lettres des clés pourront être stockées dans la table d'index. Ceci contribuera à réduire à la fois la place mémoire occupée et les temps de comparaison de la clé cherchée avec les clés de la table. Mais en cas d'égalité entre les premières lettres de différentes clés, plusieurs accès disques seront éventuellement nécessaires.

Lorsque les enregistrements physiques (repérés par un numéro) comportent plusieurs enregistrements logiques ayant chacun leur clé, on peut coder dans la table d'index, l'ensemble n° physique et position dans l'enregistrement physique de la façon suivante :

X=(*No physique**NLOG)+POSITION

où NLOG représente le nombre d'enregistrements logiques par enregistrement physique.

Le décodage se fera par :

No physique=INT(X/NLOG) POSITION=X MOD NLOG

Si la mise à jour de la table en temps réel, en cas d'ajout de clé, n'est pas programmée, c'est seulement à la prochaine constitution de la table, c'est-à-dire à la prochaine session, que les clés ajoutées y seront présentes.

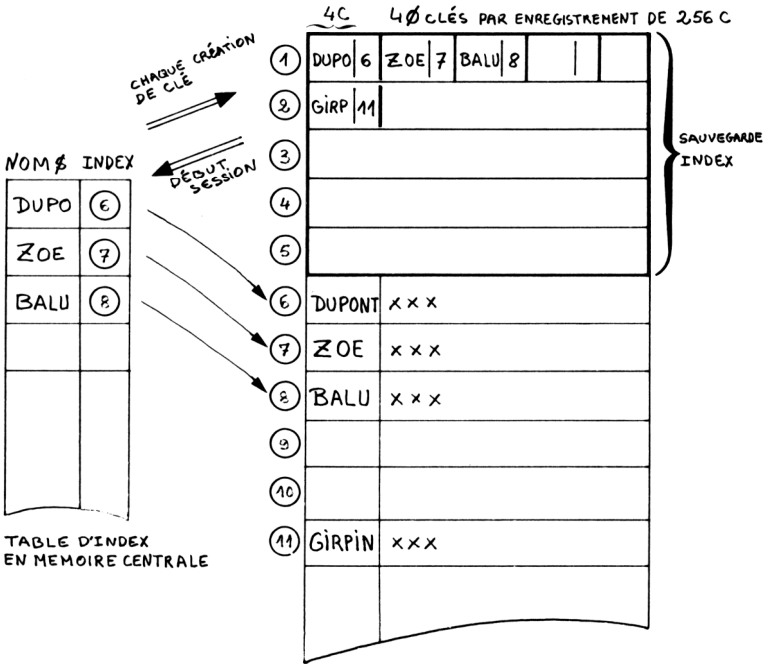
```
10 PRINT "IDO le 23.10.79":PRINT
20 '
30 '       ACCES INDEXE avec TABLE d'INDEX en MEMOIRE CENTRALE
40 '                      CONSTITUEE en DEBUT de SESSION
50 '                      ============================
70 '
80 OPEN "R",1,"CLIEN"
90 FIELD #1,10 AS N$,8 AS PN$,20 AS TL$,4 AS MT$
100 '
110 DIM NOM$(200),INDEX(200):GOSUB 190      ' Appel constitution de l'index
120 '
130 PRINT:INPUT "       Mode? (RNI) ";MODE$
140 IF MODE$="RNI" THEN GOSUB 300
150 GOTO 130
160 '
170 '-----------------------------------------------------------------------
180 '                                                  CONSTITUTION D'UN INDEX
190 J=1
200 FOR I=1 TO LOF(1)-1                     ' Lecture de tout le fichier
210    GET #1,I
220    IF ASC(N$)=0 GOTO 240                ' Enregistrement vide?
230    NOM$(J )=N$:INDEX(J)=I:J=J+1         ' Ajout de la clé dans l'index
240 NEXT I
250 '
260 RETURN
270 '
280 ----------------------------------------------------------------------
290 '                                                       RECHERCHE INDEXEE
300 PRINT: INPUT "Nom cherché? ";NOM$:IF NOM$="" THEN RETURN
310 '
320 L=LEN(NOM$)                             ' L'opérateur n'entre que les
330 '                                         premières lettres du NOM
340 FOR I=1 TO 200                          ' Lecture de la table d'index
350    IF NOM$=LEFT$(NOM$(I),L) THEN X=I:I=200:NEXT I:GOTO 410
360    IF NOM$(I)="" THEN PRINT:PRINT "N'existe pas":PRINT:GOTO 300
370 NEXT I
400 '
410 GET #1,INDEX(X)                         ' Le NOM est trouvé
420 PRINT:PRINT N$,PN$,TL$
430 GOTO 300
440 '-----------------------------------------------------------------------
460 '                  .Avec MICROSOFT 5. voir LOF:
470 '
480 '                  .SUR TRS80 NEWDOS:
490 '                                    CLEAR(3000)
500 '                                    REMPLACER LOF(1) par LOF(1)+1
510 '                                    Avant INPUT "xxxx? ";X$  faire X$=""
520 '                                    OPEN "R","TOTO:1"   (pour unité No 1)
530 '                                    PENSER à CLORE les FICHIERS
```

ACCES INDEXE AVEC SAUVEGARDE SUR DISQUE DE LA TABLE D'INDEX

Une sauvegarde de la table d'index dans un fichier et sa relecture en début de session évite la perte de temps de sa constitution mais la sauvegarde de la table d'index doit être faite à chaque création de clé et non pas seulement en fin de session; en effet, sans cette sauvegarde, si une session était interrompue, l'index sur disque ne serait plus à jour pour les sessions suivantes.

Si on a choisi d'avoir cette table en ordre croissant afin d'y faire une recherche plus rapide par dichotomie, il faut, bien sûr, lors d'une création de clé, insérer celle-ci dans la table après avoir décalé toutes les clés en aval de l'insertion, ce qui peut être relativement long.

(Voir programme ID1)



```
10 PRINT "ID1 le 23.10.79":PRINT
20 '
30 '         ACCES INDEXE  avec SAUVEGARDE de la TABLE d'INDEX
40 '         ==================================================
50 '
60 '                                          .INDEX EN MEMOIRE CENTRALE
70 '                                           SAUVEGARDE SUR DISQUE
80 '
90 NCLES=20:LCLES=4                          ' Sur TRS80 NCLES=40
100 DIM CLE$(NCLES ),PT$(NCLES)              ' LCLES= longueur clé
110 OPEN "R",1,"NOM"                         ' NCLES=nombre de clés par erg
120 '
130 FIELD #1,10 AS N$,8 AS PN$,20 AS TL$,4 AS MT$
140 FOR I=1 TO NCLES:FIELD #1,6*(I-1) AS D$,LCLES AS CLE$(I),2 AS PT$(I):NEXT
I
150 DIM NOM$(200),INDEX(200)
160 '
170 GET #1,1:IF ASC(CLE$(1))=0 THEN GOSUB 960' Appel initialisation index
180 GOSUB 500                                ' Appel lecture index
190 PRINT TAB(20) "Modes:":PRINT
200 PRINT TAB(30) "CREAT : Création"
210 PRINT TAB(30) "RNI   : Recherche par NOM Indéxée"
220 PRINT
230 PRINT:INPUT "      Mode? (CREAT,RNI,FIN) ";MODE$
240 IF MODE$="CREAT" THEN GOSUB  310
250 IF MODE$="RNI"   THEN GOSUB  900
260 IF MODE$="FIN"   THEN CLOSE #1:STOP
270 GOTO 230
280 '
290 '-----------------------------------------------------------------------
300 '                                                                CREATION
310 GOSUB 740:ON Q  GOTO 320,330,460         ' Appel recherche clé
320 PRINT "Existe deja":GOTO 310             ' Q=1 : la clé existe
330 ARANG=LOF(1):GET #1,ARANG                ' Ajout en fin de fichier
340 LSET N$=NOM$                             ' Transfert de NOM$ dans N$
350 '
360 INPUT "Prénom?    ";PR$:LSET PN$=PR$
370 INPUT "Téléphone? ";TEL$:LSET TL$=TEL$
380 INPUT "Matricule? ";MAT:LSET MT$=MKS$(MAT)
390 '
400 PRINT :PRINT TAB(30):R$="":INPUT "OK? ";R$:IF R$><"O" GOTO 320
410 PUT #1,ARANG
420 PRINT:PRINT TAB(30) "J'ai RANGE votre NOM en:";ARANG:PRINT
430 NOM$(IND)=NOM$:INDEX(IND)=ARANG:IND=IND+1 ' MAJ de la table d'index
440 GOSUB 1010                                ' Appel sauvegarde index
450 GOTO 310
460 RETURN
470 '-----------------------------------------------------------------------
480 '                                                   LECTURE DE L'INDEX
490 '                                                   EN MEMOIRE CENTRALE
500 IND=1                       ' IND:pointeur table d'index
510 FOR I=1 TO 5                ' 5 enregistrements pour l'index
520     GET #1,I
530 '
540     FOR J=1 TO NCLES
550        IF ASC(CLE$(J))=0 GOTO 590
560        NOM$(IND)=CLE$(J):INDEX(IND)=CVI(PT$(J)):IND=IND+1
570     NEXT J
580 '
590 NEXT I
600 RETURN
610 ---------------------------------------------------------------------
620 '
630 '
640 '
```

```
650 '
660 '
670 '
680 '
690 '
700 '                                                    RECHERCHE CLE
710 '                                                    Q=1 :clé trouvée
720 '                                                    Q=2 :clé n'existe pas
730 '
740 PRINT:NOM$="":INPUT "Nom?        ";NOM$:IF NOM$="" THEN Q=3:RETURN
750 '
760 L=LEN(NOM$):W=1                          ' L'opérateur n'entre que les
770 '                                        ' premières lettres du NOM
780 FOR I=W TO 200                           ' Consultation de la table d'index
790    IF NOM$(I)="" GOTO 830
800    IF NOM$=LEFT$(NOM$(I),L)  OR NOM$(I)=LEFT$(NOM$,LCLES) THEN GOTO 850
810 NEXT I
820 '
830 Q=2:RETURN
840 '                                        ' Une clé de l'index égale au
850 GET #1,INDEX(I)                          ' nom cherché a été trouvée
860 IF NOM$=LEFT$(N$,L) THEN ARANG=INDEX(I):Q=1:RETURN
870 W=I+1:GOTO 780                           ' Nom cherché>< Nom dans le fichier
880 '-----------------------------------------------------------------------
890 '                                                       RECHERCHE INDEXEE
900 GOSUB 740:ON Q GOTO 910,920,930          ' Appel recherché clé
910 PRINT:PRINT N$,PN$,TL$:GOTO 900          ' La clé existe
920 PRINT:PRINT "N'EXISTE PAS ":GOTO 900     ' La clé n'existe pas
930 RETURN
940 '-----------------------------------------------------------------------
950 '
960 PRINT :PRINT "INITIALISATION INDEX":PRINT      'INITIALISATION DE L'INDEX
970 FOR I=1 TO 5:GET #1,I:PUT #1,I:NEXT I          'avec valeurs ASCII nulles
980 RETURN
990 '-----------------------------------------------------------------------
1000 '                                                        SAUVEGARDE INDEX
1010 K=1
1020  FOR I=1 TO 5                  ' 5 enregistrements pour l'index
1030  GET #1,I
1040 '
1050     FOR J=1 TO NCLES
1060        IF NOM$(K)="" THEN PUT #1,I:RETURN
1070        LSET CLE$(J)=NOM$(K):LSET PT$(J)=MKI$(INDEX(K)):K=K+1
1080     NEXT J
1090 '
1100 PUT #1,I
1110 NEXT I
1120 '-----------------------------------------------------------------------
1130 '                  .Avec MICROSOFT 5.  voir LOF
1140 '
1150 '                  .SUR TRS80 NEWDOS:
1160 '
1170 '                                     Remplacer LOF(1) par LOF(1)+1
1180 '                                     CLEAR(3000)
1190 '                                     Avant INPUT "xxxx? ";X$  faire X$=""
1200 '                                     OPEN "R","TOTO:1"  (pour unité No 1)
```

INDEX CONTINU SUR DISQUE EN ORDRE CROISSANT

La présence en mémoire centrale d'une table d'index de taille importante peut être évitée de la façon suivante :

- Les clés sont rangées en ordre croissant dans un fichier index et une table d'index en mémoire centrale, de taille réduite, contient la première clé de chaque enregistrement du fichier index. Lors de la recherche d'une clé, une première opération permet de trouver l'enregistrement de l'index où celle-ci est située.
- Après avoir retrouvé la clé dans le fichier index, il est possible d'accéder à l'enregistrement associé à cette clé. Deux accès disques ont donc été nécessaires pour accéder à l'enregistrement cherché.
- De façon à maintenir le fichier index en ordre croissant, chaque nouvelle clé est insérée directement à sa place après avoir décalé toutes les clés en aval de l'insertion. Aussi le temps nécessaire pour un ajout de clé peut il être long lorsque le nombre de clés devient important. (quelques secondes). Si on considère que les ajouts de clés sont généralement beaucoup plus rares que les consultations, ceci est acceptable.
- Notons que le temps d'insertion d'une clé sera d'autant plus court que les enregistrements du fichier index seront plus longs puisque pour un même nombre de clés, il y aura moins d'enregistrements à décaler.

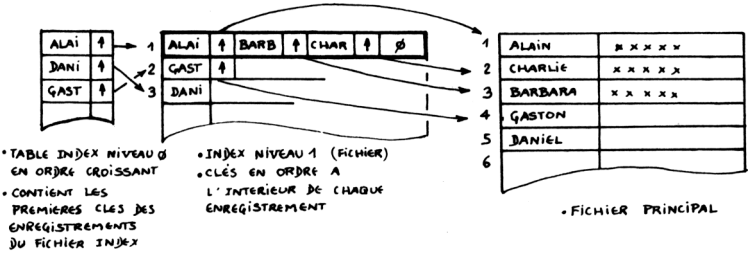
INDEX DEUX NIVEAUX

Plutôt que d'insérer des clés dans un fichier index continu, il est possible d'allouer les enregistrements de cet index dynamiquement.

A la création de l'index n'existe qu'un index de niveau 0. Un index de niveau 1, constitué dynamiquement au fur et à mesure des ajouts de clés, doit être vu comme un index continu en ordre croissant (mais, physiquement, c'est seulement à l'intérieur de chaque enregistrement de l'index 1 que l'ordre croissant des clés doit être maintenu).

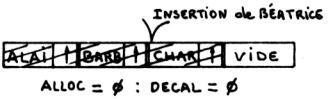
Lorsqu'une insertion est faite dans un bloc (enregistrement) plein suivi d'un autre bloc plein, un nouveau bloc, qui reçoit la dernière clé du premier bloc plein est alloué. La "continuité" de l'index 1 est assurée grâce à l'index 0. Tout se passe comme si nous avions inséré dans un index continu des "trous" qui seront utilisés lors d'insertions ultérieures de clés.

Ainsi il n'y a de décalages de clés qu'entre 2 blocs tout au plus. Pour des enregistrements de 256 caractères, l'index pourra contenir jusqu'à 40x40=1600 clés.

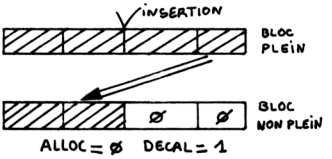


Différents cas d'insertions de clés: ALLOC → ALLOCATION D'UN NOUVEAU BLOC
DECAL → DECALAGE ENTRE 2 BLOCS

- Il y a une place libre dans le bloc de l'index où l'on insère :



- Il y a une place libre dans le bloc suivant

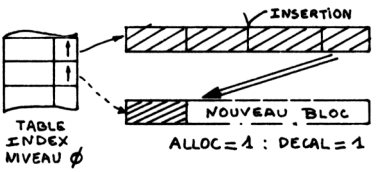


- Il n'y a pas de place libre dans le bloc suivant

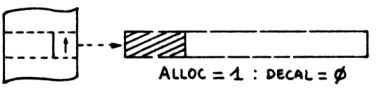
⇒ insertion dynamique d'un nouveau bloc

+ décalage entre 2 blocs

+ mise à jour de la table index de niveau Ø



- Ajout de bloc simple



ACCES INDEXE TRS80

Cet index, relativement simple à mettre en oeuvre, a pour inconvénient d'être limité en capacité si les enregistrements ont une longueur de 255 caractères au maximum (env. 300 noms).

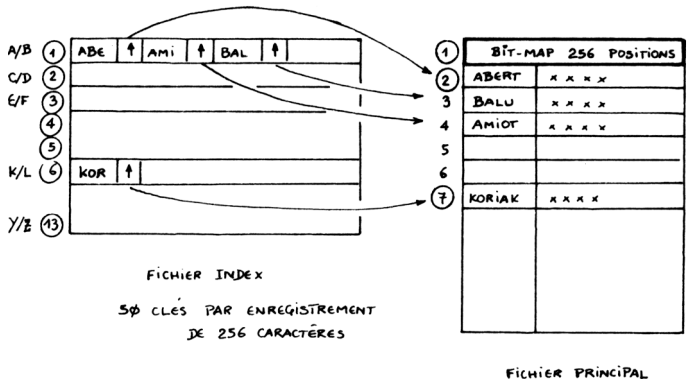
L'allocation dans le fichier index de 2 enregistrements pour chaque groupe de 2 lettres permettrait de doubler la capacité.

L'allocation dynamique des enregistrements du fichier, grâce à une pseudo "Bit-Map", évite d'avoir à réorganiser périodiquement le fichier lorsque les enregistrements supprimés doivent être récupérés.

Une simple lecture séquentielle du fichier index (en ordre croissant) fournira une liste triée des noms :

```
10 FOR I=1 TO 13          ' 13 enregistrements dans l'index
20    GET #2,I
                                          ' MXMUN:nombre de clés
30       FOR J=1 TO MXMUN
40       IF CVI(PT$(J))=0 THEN GOTO  80   ' Pointeur vide?
50       GET #1,CVI(PT$(J))
60       PRINT N$
70    NEXT J
80 NEXT I
```

Remarquons qu'un index détruit peut être regénéré simplement par la lecture séquentielle du fichier principal et par l'appel des sous-programmes de recherche et d'ajout de clé déjà écrits. Une Bit-Map se regénère également simplement.



FICHIER INDEX

5Ø CLÉS PAR ENREGISTREMENT
DE 256 CARACTÈRES

FICHIER PRINCIPAL

```
10 ' ID5
20 '
30 ' INDEX   TRS80  (1 ENREGISTREMENT LOGIQUE PAR ENREGISTREMENT  PHYSIQUE)
40 '
50 ' ALLOCATION DYNAMIQUE des ENREGISTREMENTS du FICHIER PRINCIPAL
60 '                              (Gestion par BIT-MAP)
70 '
80 MXMUN=25: TENREG=128                  ' Faire MXMUN=50:TENREG=254 sur TRS80
90 DIM PT$(MXMUN),CLE$(MXMUN),BMAP$(TENREG)
100 OPEN "R",1,"TOTUS":OPEN "R",2,"ITOTU"
110 FIELD #1,12 AS N$                                          '     Voir dessin
120 FOR I=1 TO TENREG:FIELD #1,(I-1) AS D$,1 AS BMAP$(I):NEXT I
130 FOR I=1 TO MXMUN:FIELD #2,5*(I-1) AS D$,2 AS PT$(I),3 AS CLE$(I):NEXT I
140 '
150 INPUT "NOM? ";NOM$:IF NOM$="FIN" THEN CLOSE #1,#2:STOP  ' Sequence d'essai
160 '
170 GOSUB 500:IF Q><1 THEN GOTO 150 ELSE GOSUB 220:ON Q GOTO 180,200,150
180 PRINT N$:GOTO 150
190 '
200 INPUT "Nouveau NOM  OK? ";R$:IF R$><"O" GOTO 150 ELSE LSET N$=NOM$:
             PUT #1,ARANG:GOSUB 430:GOTO 150
210 '-----------------------------------------------------------------------
220 GET #2,IX                                                  '    RECHERCHE CLE
230 FOR M=1 TO MXMUN
240    IF CVI(PT$(M))=0 THEN FR=M:GOTO 270
250 NEXT M
260 '
270 FOR M=1 TO MXMUN                  '#1 ---> FICHIER PRINCIPAL
280    IF CVI(PT$(M))=0 GOTO 390      'ARANG -->ADRESSE ENREGISTREMENT du  FP
290    IF CLE$<CLE$(M)  GOTO 390      'IX --> ADRESSE FICHIER INDEX
300    IF CLE$=CLE$(M) GOTO 340
310 NEXT M
320 '
330 IF CVI(PT$(M))=0 GOTO 390
340 ARANG=CVI(PT$(M)):GET #1,ARANG
350 IF NOM$<LEFT$(N$,LEN(NOM$)) GOTO 390
360 IF LEFT$(N$,LEN(NOM$))=NOM$ THEN
        INSER=M: Q=1:RETURN                    ' Cle trouvee Q=1
370 M=M+1:GOTO 330
380 '
390 INSER=M:GOSUB 640:GET #1,ARANG:Q=2         ' Cle non trouvee Q=2
400 IF CVI(PT$(MXMUN-1))><0 THEN PRINT "INDEX PLEIN":Q=3:RETURN
410 RETURN
420 '-----------------------------------------------------------------------
430 GET #1,1:LSET BMAP$(ARANG)=CHR$(1):PUT #1,1                '        AJOUT CLE
440 FOR U=FR-1 TO INSER STEP -1:
           LSET CLE$(U+1)=CLE$(U):LSET PT$(U+1)=PT$(U):NEXT U
450 LSET PT$(INSER)=MKI$(ARANG):LSET CLE$(INSER)=CLE$
460 PUT #2,IX
470 RETURN
480 '-----------------------------------------------------------------------
490 '                                                  ' CALCUL ADRESSE INDEX
500 CLE$=LEFT$(NOM$,3)                                     ( 1<=IX<=13 )
510 IX=ASC(LEFT$(NOM$,1))-64
520 IX=INT((IX+1)/2):Q=1:IF IX<1 OR IX>13 THEN Q=2
530 RETURN
540 '-----------------------------------------------------------------------
550 '                                 SUPPRESSION ENREGISTREMENT DE LA BIT-MAP
560 GET #1,1:LSET BMAP$(ARANG)=CHR$(0):PUT #1,1
570 FOR U=INSER TO FR-1:LSET PT$(U)=PT$(U+1):LSET CLE$(U)=CLE$(U+1):NEXT U
580  PUT #2,IX:RETURN
590 '
600 '
610 '
615 '
616 '
```

```
620 '-----------------------------------------------------------------------
630 '                                   RECHERCHE ENREGISTREMENT DANS BIT-MAP
640 GET #1,1
650 IF BMAP$(TENREG)><"*" THEN PRINT "INITIALISATION INDEX (WAIT) ":
      PRINT :GOSUB 740
660 '
670 FOR W=2 TO TENREG
680    IF ASC(BMAP$(W))=0 THEN ARANG=W:RETURN
690 NEXT W
700 '-----------------------------------------------------------------------
730 '                                                          INITIALISATION
740 FOR P=1 TO TENREG:LSET BMAP$(P)=CHR$(0):NEXT P:
            LSET BMAP$(TENREG)="*":PUT #1,1
750 FOR P=1 TO 13
760    GET #2,P:FOR Q=1 TO MXMUN:LSET PT$(Q)=MKI$(0):NEXT Q:PUT #2,P
770 NEXT P
780 RETURN
790 '-----------------------------------------------------------------------
800 '
810 '    EXEMPLE:
820 '                NOM$="MARTIN"
830 '                GOSUB 500:GOSUB 220  :ON Q GOTO X,Y
840 '            X   CLE TROUVEE
850 '
860 '            Y   :GOSUB 430           (AJOUT)
870 '
880 '
```

CHAPITRE 3-3

LES BASES DE DONNEES

L'utilisateur de P.S.I. de gestion se trouve confronté à des problèmes de consultations et de mises à jour de données en temps réel ; il est donc amené à faire une approche de son problème sous forme de base de données.

Il n'existe pas encore de Système de Gestion de Base de Données logiciel (SGBD) sur les micro-ordinateurs de gestion, ne serait-ce que pour des raisons d'encombrement en mémoire centrale. D'ailleurs, peut-être ne sont-ils pas souhaitables ; en effet, si le SGBD soulage le programmeur de la gestion des pointeurs, de la gestion des sécurités contre les destructions physiques et définit un langage commun, il peut aussi être d'une utilisation moins souple que des fichiers élaborés (indexés avec allocation dynamique).

Qui dit base de données dit pointeurs entre enregistrements de plusieurs fichiers. Pointeurs que l'utilisateur devra gérer lui-même et contrairement à une idée répandue, ceux-ci se gèrent sinon simplement, du moins avec peu d'instructions.

- Pourquoi les bases de données.

- Gestion pointeurs.

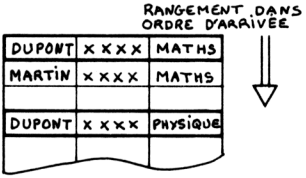
- Sécurités contre les destructions physiques de fichiers.

- Exemple de base de données : gestion de commandes et livraisons clients-fournisseurs.

POURQUOI LES BASES DE DONNEES

Soit à gérer des instructions à des cours : supposons que ces inscriptions soient enregistrées dans un fichier au fur et à mesure de leur arrivée. *(figure 1)*

Il n'est pas possible avec ce fichier de vérifier immédiatement avant d'inscrire une personne à un cours s'il reste des places disponibles pour ce cours, et si cette personne n'est pas déjà inscrite à ce cours ; en effet, l'exploration d'un fichier de 1 000 enregistrements par exemple demande 50 secondes si le temps d'accès à un enregistrement est de 50 millisecondes.



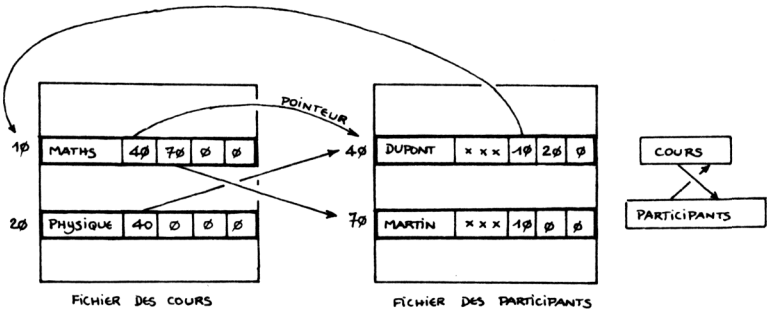
INSCRIPTION A DES COURS

De même, la recherche de tous les participants inscrits à un cours déterminé ne peut pas se faire en moins de 50 secondes.

En outre, les renseignements concernant un participant sont saisis et stockés autant de fois que celui-ci est inscrit à différents cours.

Rendons indépendants les participants des cours en éclatant le fichier précédent en un fichier des participants et un fichier des cours. *(figure 2)*

Nous pouvons ranger les participants ainsi que les cours en fin de fichier et prévoir un index pour y accéder directement.



Etablissons ensuite les liens entre les enregistrements de ces deux fichiers :

- Dans chaque enregistrement du fichier cours, indiquons les numéros des enregistrements des participants inscrits au cours correspondant (pointeurs).

- De même notons dans le fichier des participants les numéros des enregistrements des cours suivis par chaque participant.

On peut maintenant, lors de l'inscription d'un participant à un cours, vérifier si ce participant existe déjà, s'il

reste des places disponibles pour ce cours et s'assurer grâce aux pointeurs que ce participant n'est pas déjà inscrit à ce cours.

Une liste de tous les participants à un cours peut être obtenue immédiatement à l'aide des pointeurs.

Chaque "pointeur" ne nécessite que peu de place (2 octets permettant de numéroter 64 000 enregistements dans les fichiers).

Il est possible d'indiquer dans les enregistrements des cours non plus les numéros des enregistrements des participants mais directement les noms de ceux-ci. *(figure 3)* On parle alors de pointeurs logiques qui ont l'avantage de rendre le fichier des cours physiquement indépendant du fichier des participants ; en effet, si un participant qui aurait été rangé dans un fichier à accès direct est déplacé à l'occasion d'une réorganisation de ce fichier, il n'y a pas à modifier les pointeurs correspondants dans le fichier des cours. Par contre, ces pointeurs logiques exigent beaucoup plus de place (10 octets au lieu de 2 par exemple), et il faudra systématiquement recourir à l'index pour retrouver les participants.

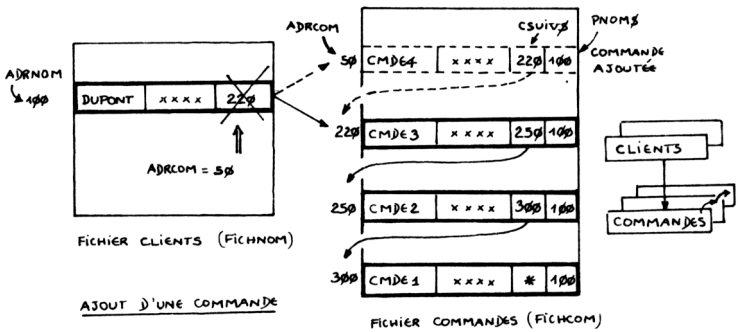
POINTEURS "LOGIQUES"

GESTION DES POINTEURS

Ajouts d'éléments dans une chaîne

Soient un fichier de clients et un fichier de commandes. Si chaque client a plusieurs commandes que l'on désire retrouver immédiatement, un chaînage de celles-ci entre elles, au fur et à mesure de leur arrivée, est nécessaire *(figure 4)*.

Le plus simple est d'ajouter toute nouvelle commande en début de chaîne.



Nous lisons d'abord en mémoire centrale l'enregistrement du client et celui de la commande à ajouter (à moins qu'ils n'y

soient déjà) puis nous procédons à la mise à jour des pointeurs CSUIVØ,PCOMØ et PNOMØ.

```
GET FICHNOM,ADRNOM          ' Lire d'abord les ENREGISTREMENTS
GET FICHCOM,ADRCOM          '            en MEMOIRE CENTRALE

------------------------
 LSET CSUIV$=PCOM$          ' Transférer pointeur PCOM$ dans pointeur CSUIV$
 LSET PCOM$=MKI$(ADRCOM)    ' Transférer adresse nouvelle commande dans PCOM$
 LSET PNOM$=MKI$(ADRNOM)    ' Transférer adresse client dans PNOM$
------------------------    '

PUT FICHNOM,ADRNOM          ' Ranger le client
PUT FICHCOM,ADRCOM          ' Ranger la commande
```

Remarques : A la création d'un nouveau client, le pointeur "PCOMØ" est initialisé par LSET PCOMØ="*" . L'insertion dans une "chaîne" peut être faite de façon à ce que les éléments soient dans un certain ordre.

Lecture d'une chaîne : Pour retrouver toutes le commandes d'un client, dont nous connaissons l'adresse de rangement ADRNOM, nous programmons la séquence suivante :

```
     GOSUB LECTURE:STOP                  ' On connait ADRNOM

LECTURE:GET FICHNOM,ADRNOM               ' Lecture du client
        IF PCOM$="* " THEN RETURN        ' Chaîne  vide ?
        COM=CVI(PCOM$)
    BCL:GET FICHCOM,COM:PRINT NUMCOM$    ' NUMCOM$=Zone numéro de commande
        IF CSUIV$="* " THEN RETURN       ' Fin ?
        COM=CVI(CSUIV$):GOTO BCL         ' Commande suivante
```

Exemple : Séquence pour une commande de matériels

Regardons sur une séquence complète d'une commande de matériels comment va s'effectuer le chaînage des commandes d'un CLIENT.

Tout d'abord on RECHERCHE le CLIENT : pour cela, un sous-programme chargé de cette recherche est appelé, et si le client n'existait pas, de le créer, sans qu'il soit nécessaire de passer par un 'mode création'. Le pointeur 'PCOMØ' vers les commandes étant dans ce cas initialisé avec "*" (fin de chaîne).

Le sous-programme CLIENT appelle lui-même un sous-programme CAL-ADRNOM chargé de rechercher l'enregistrement où se trouve le client si celui-ci existe déjà, sinon de rechercher un enregistrement libre où le ranger. Ce dernier sous-programme positionne un indicateur Q soit à 1 si le client existe, soit à 2 s'il n'existe pas, permettant ainsi au sous-programme appelant CLIENT, par le test de Q, de savoir s'il doit faire la saisie du client.

Donc, au retour de ce sous-programme 'CLIENT', le client existera de toute façon (sauf si l'on avait répondu par un RC à la question 'CLIENT ?' pour signifier un changement de mode par le positionnement d'un indicateur Q à 2 analyse au retour du sous-programme).

Ce sous-programme CLIENT peut être appelé par une autre partie du programme. Par exemple un mode qui permet de créer

un nouveau client. On remarque que ce mode ne nécessite que 2 instructions.

```
-------------------------------------------------------
CREACLIENT: GOSUB CLIENT : ON Q GOTO CREACLIENT,DEBUT

        .Q=2 l'utilisateur veut changer de mode
-------------------------------------------------------
```

Les coordonnées d'un client doivent pouvoir être modifiées. Aussi la séquence correspondant à la saisie du client sera-t-elle transformée en un sous-programme que l'on appelera également d'un mode 'MODIFICATION CLIENT'.

Appelons maintenant un sous-programme CAL-ADRCOM chargé de rechercher un enregistrement libre dans le fichier des commandes où la commande peut être rangée. Si celle-ci existe déjà, ce sous-programme positionne un indicateur Q à 1 et au retour de ce sous-programme nous refusons bien sûr d'entrer une commande qui existe déjà.

Saisissons ensuite la commande elle-même.

- Pour cela nous allons consulter un fichier des matériels. Là encore, c'est un sous-programme qui est chargé de retrouver un matériel et de le créer s'il n'existe pas (sans changer de mode).

- Ce sous-programme pourra aussi servir à la saisie d'un nouveau matériel indépendamment de la saisie d'une commande.

Avant d'écrire effectivement la commande dans le fichier, nous demandons à l'opérateur s'il est d'accord.

COMMANDES de MATERIELS
==========

```
DEBUT:   INPUT "Mode? ";MODE$
         IF MODE$="COMMANDE" GOTO COMMANDE
         IF MODE$="        " GOTO

             !
             !
             V
         GOTO DEBUT

COMMANDE: GOSUB CLIENT: ON Q GOTO 1,DEBUT

       1: INPUT "COMMANDE? ";COM$
          IF COM$="" GOTO COMMANDE
          GOSUB CAL-ADRCOM: ON Q GOTO 2,3

       2: PRINT "Com existe déjà": goto 1
       3: INPUT "Date? ";

                .
                .

       4: GOSUB MATERIEL: ON Q GOTO 5,6

       5: INPUT "Combien? "  'plusieurs
          INPUT "Delai  ? "  'matériels
          INPUT "Prix   ? "  'par commande

          SEQUENCE NON ETUDIEE

          GOTO 4

       6: INPUT "Commande OK? ";RP$
          IF RP$><"O" GOTO COMMANDE

          ----------------------
          LSET CSUIV$=PCOM$
          LSET PCOM$=MKI$(ADRCOM)   pointeurs
          LSET PNOM$=MKI$(ADRNOM)
          ----------------------
          --------------------
          PUT FICHNOM,ADRNOM
          PUT FICHCOM,ADRCOM
          --------------------

          GOTO COMMANDE
```

Sous-Programme chargé de RECHERCHER un CLIENT: Le CREE s'il n'EXISTE PAS

```
RETOUR: Q=1 OK
        Q=2 FIN

CLIENT: INPUT "Client? ";CLIENT$
        IF CLIENT$="" THEN Q=2:RETURN

        GOSUB CAL-ADRNOM:ON Q GOTO 1,2

      2: INPUT "Nouv CLIENT ? ";RP$
         IF RP$><"O" GOTO CLIENT

         INPUT "Adresse ? "
         INPUT "Teleph  ? "

         ------------------------
         LSET PCOM$="*"  <-- initialisation du pointeur
         PUT FICHNOM,ADRNOM
         ------------------------

      1: Q=1:RETURN
```

Sous-Programme chargé de RECHERCHER un MATERIEL :Le CREE s'il n'EXISTE PAS

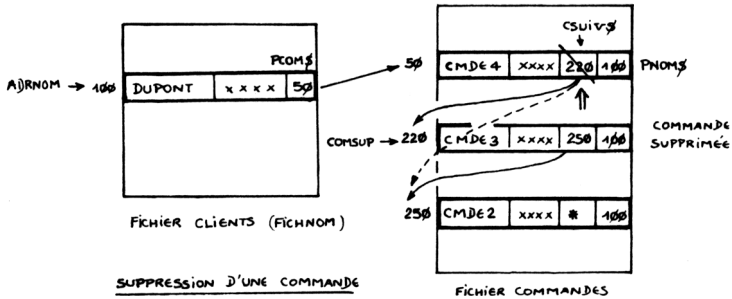
```
RETOUR: Q=1 OK
        Q=2 FIN

MATERIEL: INPUT "Matériel? ";MATER$
          IF MATER$="" THEN Q=2:RETURN

          GOSUB CAL-ADRMAT:ON Q GOTO 1,2

        2: INPUT "Nouv MATERIEL ? ";RP$
           IF RP$><"O" GOTO MATERIEL

           INPUT "Libellé ? ";
           INPUT "Prix    ? ";

        1: Q=1:RETURN
```

==========
Remarquez les indicateurs 'Q' positionnés par les sous-programmes et analyses au retour par:

ON Q GOTO xx,xx
==========

Suppressions d'éléments dans une chaîne (figure 6)

Supposons que nous voulions supprimer une commande à laquelle nous venons d'accéder directement (par un index par exemple). Le pointeur de la commande précédente (dans la chaîne) doit être mis à jour. Pour retrouver celle-ci, nous nous positionnons d'abord sur le client (grâce au pointeur PNOM$) puis nous explorons la chaîne jusqu'au moment où nous retrouvons la commande à supprimer. Nous procédons alors à la mise à jour.



SUPPRESSION D'UNE COMMANDE

```
     GOSUB SUP:STOP          ' On connait l'adresse de la commande
                               à supprimer (COMSUP).
     ADRNOM=CVI(PNOM$)       ' Calcul de ADRNOM
 SUP:GET FICHNOM,ADRNOM
     IF PCOM$="* " THEN PRINT "Chaîne vide":Q=3:RETURN
     IF CVI(PCOM$)=COMSUP THEN GET FICHCOM,CVI(PCOM$):LSET PCOM$=CSUIV$:
          PUT FICHNOM,ADRNOM:Q=1:RETURN
                                                 ' C'était la 1ère commande
     X=CVI(PCOM$)            ' Adresse première commande

BCL: GET FICHCOM,X
     IF CVI(CSUIV$)=COMSUP THEN  GET FICHCOM,CVI(CSUIV$):Y$=CSUIV$:
                GET FICHCOM,X:LSET CSUIV$=Y$:PUT FICHCOM,X:Q=1:RETURN

     IF CSUIV$="* " THEN PRINT "Commande non trouvée":Q=2:RETURN

     X=CVI(CSUIV$):GOTO BCL
```

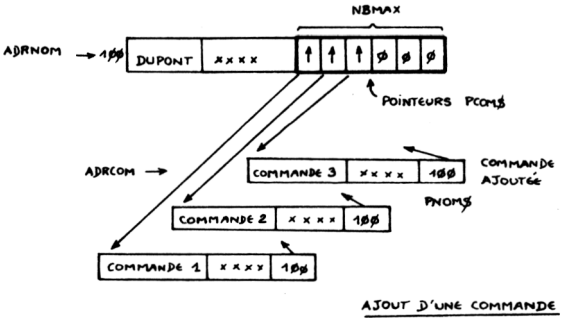
Si la commande sur laquelle nous étions positionné, au départ, n'est pas retrouvée par le chaînage, nous signalons l'anomalie à l'opérateur.

Un chaînage détruit peut être régénéré en recherchant toutes les commandes qui "pointent" vers le même nom.

Remarquons qu'avec un "chaînage arrière", la commande précédant la commande à supprimer, serait retrouvée directement sans lire toute la chaîne.

S'il est possible de connaître le nombre maximum de commandes pour un client, (5 par exemple), il est réservé autant de

pointeurs qu'il peut y avoir de commandes. La gestion des pointeurs devient dans ce cas sensiblement plus simple pour les suppressions *(figure 7)*.



Ajout

Les enregistrements d'adresse ADRNOM et ADRCOM sont en mémoire centrale

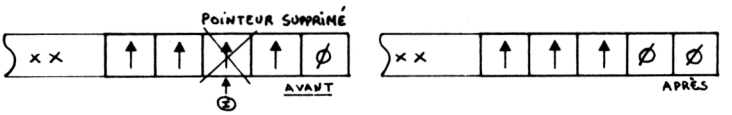
```
      GOSUB AJOUT:STOP

AJOUT:FOR I=1 TO NBMAX-1                (Recherche d'un pointeur libre)
        IF CVI(PCOM$(I))=0 THEN
              LSET PCOM$(I)=MKI$(ADRCOM):LSET PNOM$=MKI$(ADRNOM):
              PUT FICHNOM,ADRNOM:PUT FICHCOM,ADRCOM:Q=1:RETURN
      NEXT I

      PRINT "Il n'y a plus de pointeur libre":Q=2:RETURN
```

Suppression

L'adresse de la commande à supprimer COMSUP est connue. On lit d'abord l'enregistrement du client dont l'adresse nous est donnée par PNOM$. Puis le pointeur PCOM$ correspondant à la commande à supprimer est effacé.

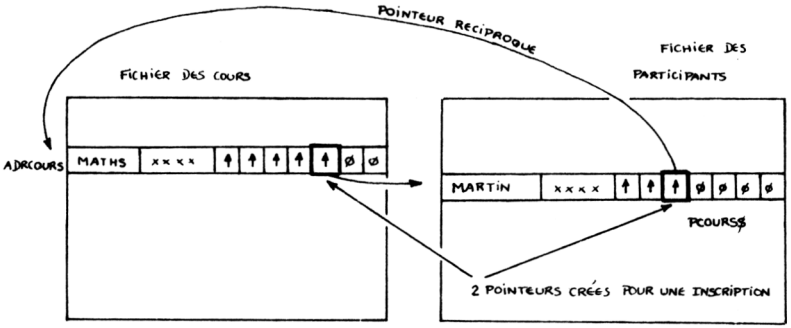
```
       GOSUB SUP:STOP                 ' On connait COMSUP

       ADRNOM=CVI(PNOM$):GET FICHNOM,ADRNOM
SUP:   FOR I=1 TO NBMAX-1
       IF CVI(PCOM$(I))=COMSUP THEN FOR V=I TO NBMAX-1:
         LSET PCOM$(I)=PCOM$(I+1):NEXT V:PUT FICHNOM,ADRNOM:Q=1:RETURN
       NEXT I

       PRINT "Pointeur non trouvé":Q=2:RETURN
```

Le "tassement" des pointeurs, de façon à ne pas avoir de "trou", n'est nécessaire que si les commandes doivent être retrouvées dans leur ordre d'arrivée.



SUPPRESSION D'UNE COMMANDE

Sécurités pointeurs (figure 9)



Soient un fichier des cours et un fichier des participants à ces cours. Lorsqu'un participant est inscrit à un cours, 2 pointeurs sont créés :

1 pointeur cours --→ participant
1 pointeur participant --→ cours

Ces pointeurs permettront de retrouver respectivement tous les participants à un cours et tous les cours suivis par un participant.

Après avoir saisi une inscription à l'écran, nous écrivons dans 2 fichiers : le fichier des cours et celui des participants.

```
PUT FICCOURS,ADRCOURS
                                    <----- PROGRAMME INTERROMPU

PUT FICPARTI,ADRPARTI
```

Si le programme est interrompu entre les 2 écritures, seul un des 2 pointeurs est créé. Par conséquent, de façon à limiter le risque d'interruption entre 2 instructions d'écriture sur fichier, celles-ci doivent se suivre. Et, afin de détecter, malgré tout, les incohérences de pointeurs, nous vérifions lorsque nous accédons à un enregistrement d'un fichier par l'intermédiaire d'un pointeur, que dans cet enregistrement le "pointeur réciproque" vers l'enregistrement où nous avions trouvé le premier pointeur existe bien.

Par exemple, lorsque nous sommes positionnés sur un cours et que nous allons lire dans le fichier des participants ceux inscrits à ce cours, nous vérifions que chacun de ces participants possède bien un pointeur vers ce cours.

```
        GOSUB VERIF:STOP

VERIF:  FOR I=1 TO NBPOINTCOURS
        IF CVI(PCOURS$(I))=ADRCOURS THEN Q=1:RETURN
        NEXT I

        PRINT "IL MANQUE UN POINTEUR":Q=2:RETURN
```

Si ce n'est pas le cas, l'opérateur doit en être averti. La possibilité d'autocorrection de l'erreur, par ajout du pointeur manquant, aura été prévue.

Les pointeurs réciproques permettront de détecter une perte accidentelle d'un enregistrement qui ne l'aurait pas été par ailleurs (et que l'on récupèrera sur une sauvegarde).

Sécurités contre les destructions physiques de fichiers

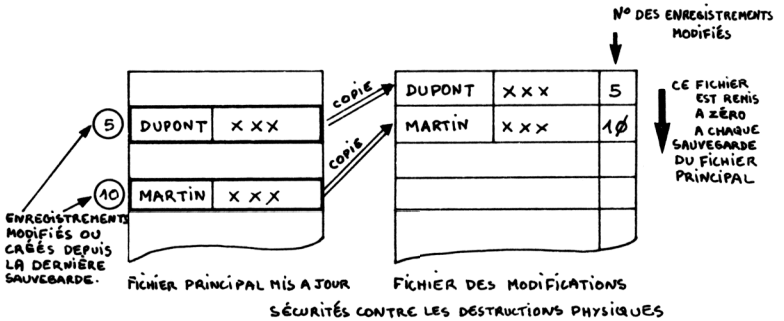
(figure 10)

Rappelons une règle qui n'est pas toujours connue :

En temps réel, des sauvegardes périodiques des fichiers, fussent-elles très rapprochées, ne sont pas suffisantes puisqu'en cas de destruction d'un fichier toutes les modifications sur celui-ci, depuis sa dernière sauvegarde sont perdues. Il est donc nécessaire de garder une trace de celles-ci dans un autre fichier (sur une autre disquette).

Ainsi, si un des fichiers mis à jour est détruit, il est régénéré grâce à sa dernière sauvegarde et aux modifications depuis cette sauvegarde. Dès que les fichiers des modifications deviennent trop importants, une nouvelle sauvegarde des fichiers est faite puis les fichiers des modifications sont "remis à zéro".

Remarque : La mise à jour d'une sauvegarde à l'aide des modifications est plus rapide qu'une sauvegarde complète.



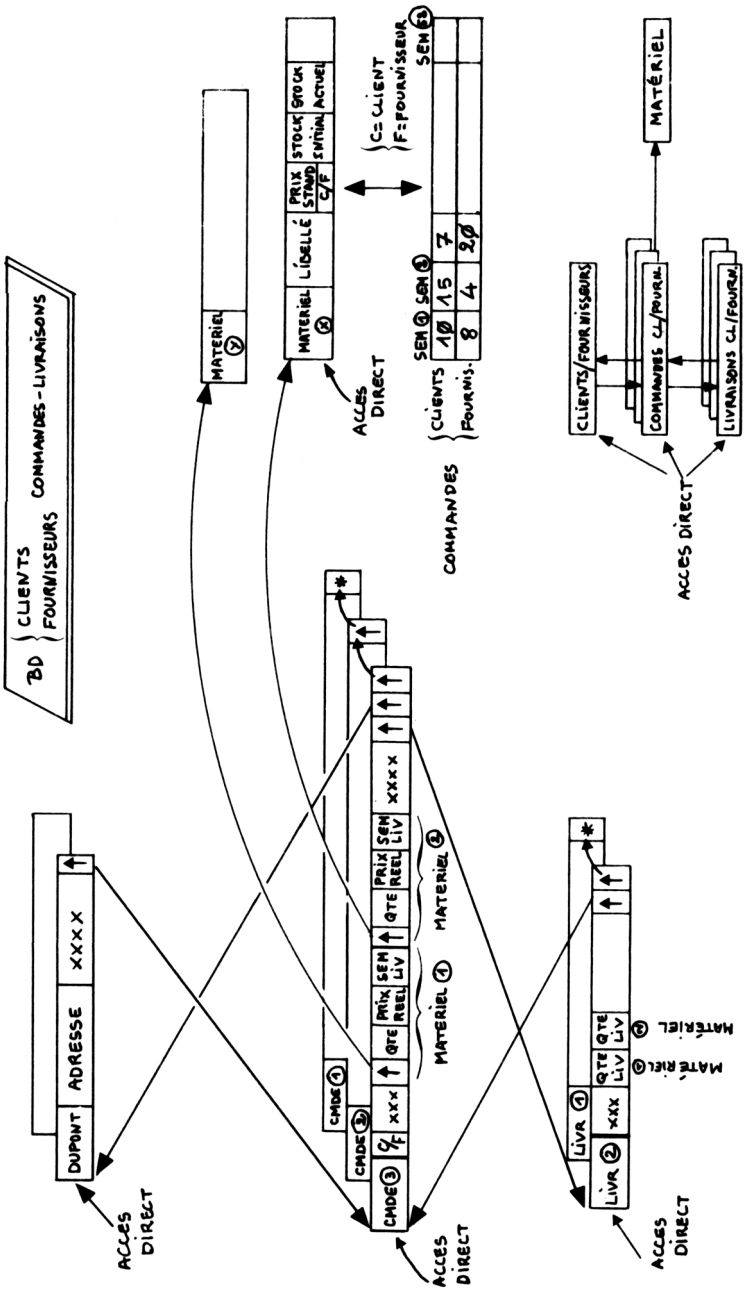
SÉCURITÉS CONTRE LES DESTRUCTIONS PHYSIQUES

EXEMPLE DE BASE : GESTION DE COMMANDES ET LIVRAISONS CLIENTS-FOURNISSEURS

Cette base de données *(figure 11)* doit permettre de gérer des commandes provenant de clients ainsi que celles émises vers des fournisseurs. La gestion des stocks peut donc être prévisionnelle.

Au moment de l'établissement d'une commande CLIENT, il est possible de savoir si un matériel est livrable pour une certaine date, en prenant en considération :

- le stock actuel
- les commandes clients
- les commandes fournisseurs.


BD
CLIENTS
FOURNISSEURS
COMMANDES - LIVRAISONS
DUPONT
ADRESSE
XXXX
ACCES DIRECT
MATERIEL
MATERIEL
LIBELLÉ
PRIX STAND
C/F
STOCK INITIAL
STOCK ACTUEL
C= CLIENT
F= FOURNISSEUR
SEM
CLIENTS
FOURNIS.
COMMANDES
CLIENTS/FOURNISSEURS
COMMANDES CL/FOURN.
LIVRAISONS CL/FOURN.
MATÉRIEL
CMDE
C/F
XXX
QTE
PRIX REEL
SEM LIV
MATERIEL
LIVR
MATÉRIEL

L'état des différentes livraisons d'une commande ainsi que le solde à livrer sont connus en temps réel.

Nous avons choisi d'avoir un accès direct sur tous les fichiers (clients, commandes, livraisons, matériel). En 'entrant' dans la base par un numéro de livraison, il est possible, grâce aux pointeurs, de 'remonter' vers la commande correspondante puis vers le client.

Cette base se constitue dynamiquement. Par exemple, lors de l'établissement d'une commande, si un matériel n'existe pas, il y a appel d'un sous-programme de création d'un nouveau matériel. Il en est de même pour les clients.

Gestion client-commandes

Nous n'avons programmé que la saisie client et la gestion du chaînage des différentes commandes de chaque client. L'accès indexé, programmé pour les clients, ne l'a pas été pour les commandes. Il doit bien sûr être ajouté si on veut accéder directement à une commande par son seul numéro sans avoir à connaître le client qui peut être ensuite retrouvé grâce au pointeur PNOM$.

```
10 PRINT "BDCOM le 1.1.80":PRINT
20 '
30 '              GESTION CLIENT-COMMANDES
40 '              ========================
50 '
60 OPEN "R",1,"CLIEN":OPEN "R",2,"COM"
70 FIELD #1,20 AS N$,25 AS RUE$,15 AS VILLE$,5 AS CPOST$,20 AS TL$,2 AS PCOM$
80 FIELD #2,12 AS NCOM$,50 AS D$,2 AS CSUIV$,2 AS PNOM$
90 '
100 DIM NOM$(200),INDEX(200)
110 GOSUB 850                          ' Appel constitution de l'index client
120 '
130 PRINT:INPUT "      Mode? (COM,LCOM,FIN )";MODE$
140 IF MODE$="COM" GOTO 200            ' Commande
150 IF MODE$="LCOM" GOTO 710           ' Liste les commandes d'un client
160 IF MODE$="FIN" THEN CLOSE #1:STOP
170 GOTO 130
180 '-----------------------------------------------------------------------
190 '                                                          MODE COMMANDE
200 GOSUB 380:ON Q GOTO 210,130        ' Appel recherche client
210 PRINT:PRINT N$:PRINT :ACOM=LOF(2):PRINT "No commande:";ACOM
220 GET #2,ACOM
230 '
240 PRINT:PRINT "   Saisie COMMANDE:":PRINT
250 PRINT :INPUT "   No Commande? ";X$:LSET NCOM$=X$
260 '
270 '   saisie commande a completer
280 '
290 PRINT:INPUT "   Commande OK? ";R$:IF R$><"O" GOTO 200
300 '
310 LSET CSUIV$=PCOM$:LSET PNOM$=MKI$(ANOM)  ' Création des pointeurs
320 LSET PCOM$=MKI$(ACOM)                    ' (chaînage)
330 '
340 PUT#1,ANOM:PUT #2,ACOM
350 GOTO 200
360 '-----------------------------------------------------------------------
370 '                                          RECHERCHE/CREATION CLIENT
380 PRINT: INPUT "Nom CLIENT? ";X$
390 IF X$="" THEN Q=2:RETURN                 ' Q=2:fin du mode
400 '
410 L=LEN(X$)
420 '
430 FOR I=1 TO 200                           ' Recherche clé dans table index
440   IF X$=LEFT$(NOM$(I),L) THEN ANOM=INDEX(I):GET #1,INDEX(I):Q=1:RETURN
450   IF NOM$(I)="" GOTO 480
460 NEXT I
470 '                                        ' Le client n'existe pas
480 PRINT :PRINT X$;:INPUT "  Nouveau CLIENT? ";RP$
490 IF RP$><"O" GOTO 380
500 '
510 ANOM=LOF(1):GET #1,ANOM:LSET N$=X$:PRINT ' Rangement en fin de fichier
520 INPUT "Rue       ? ";X$:LSET RUE$=X$     ' Saisie du nouveau client
530 INPUT "Ville     ? ";X$:LSET VILLE$=X$
540 INPUT "Code Post ? ";X$:LSET CPOST$=X$
550 INPUT "Telephone ? ";X$:LSET TL$=X$
560 '
570 LSET PCOM$="*"                           ' Initialisation pointeur PCOM$
580 PUT #1,ANOM
590 NOM$(PI)=N$:INDEX(PI)=ANOM:PI=PI+1       ' MAJ de la table d'index client
600 '
610 Q=1:RETURN
620 '
```

```
630 '
640 '
650 '
660 '
670 '
680 '
690 '
700 '-----------------------------------------------------------------------
710 GOSUB 380:ON Q GOTO 720,130            ' LISTE DES COMMANDES D'UN CLIENT
720 PRINT:PRINT N$:PRINT
730 PRINT TAB(20) RUE$
740 PRINT TAB(20) VILLE$
750 PRINT TAB(20) CPOST$ TAB(35) "Tel:";TL$
760 '
770 PRINT:IF PCOM$="* " GOTO 710           ' Y a t-il une commande?
780 COM=CVI(PCOM$)
790 '
800 GET #2,COM:PRINT "Commande:"; NCOM$
810 IF CSUIV$="* " GOTO 710                ' Dernière commande?
820 COM=CVI(CSUIV$):GOTO 800               ' Adresse commande suivante
830 '-----------------------------------------------------------------------
840 '                                      ' CONSTITUTION DE L'INDEX CLIENT
850 PI=1                                   ' PI:pointeur table d'index
860 '
870 FOR I=1 TO LOF(1)                      ' Lecture de tout le fichier
880    GET #1,I
890    IF ASC(N$)=0 GOTO 910               ' Enregistrement vide?
900    NOM$(PI)=N$:INDEX(PI)=I:PI=PI+1
910 NEXT I
920 '
930 RETURN
940 '
950 ----------------------------------------------------------------------
```

A N N E X E I

RAPPELS SUR BASIC

Connexion sous Basic

La connexion sous un système BASIC se fait généralement par une commande :

BASIC (retour chariot) pour TRS80-DTC MICROFILE

ou

MBASIC (retour chariot) pour BASIC MICROSOFT 5.

Après la connexion le système BASIC répond :

OK (OK permet de s'assurer que l'on est sous BASIC)

(sur TRS80, le système BASIC répond READY)

On peut alors utiliser le BASIC pour :

- faire directement des calculs (mode direct)
- écrire des programmes.

Mode direct

Frappons au clavier

PRINT 4 (Retour chariot)

le calculateur affiche le nombre 4.

Entrons maintenant

A=4 (Retour chariot)
B=5 (Retour chariot)

le calculateur affecte à A et B que nous appellerons variables les valeurs 4 et 5.

Si maintenant nous frappons au clavier :

PRINT A+B

le calculateur affiche 9 (la somme des valeurs des variables A et B).

Mode programme

Ecrivons maintenant un programme :

```
10 A=4
20 B=5
30 PRINT A+B
```

10, 20 et 30 représentent des numéros d'instructions.

Lorsque nous demandons au calculateur d'exécuter le programme par la commande 'RUN', les instructions s'exécutent les unes après les autres dans l'ordre de leur numérotation.

RUN (retour chariot)

9 est affiché à l'écran.

Le programme est réexécutable autant de fois que nous le désirons.

Pour le sauvegarder sur disque nous écrivons la commande :

SAVE "TOTO" (où 'TOTO' représente le NOM du PROGRAMME)

Nous pourrons, plus tard, recharger ce programme dans la mémoire centrale (où le programme peut être exécuté) par une commande :

LOAD "TOTO"

Lorsque le programme a été exécuté, nous pouvons demander en 'Mode Direct' :

PRINT AxB

Des valeurs ayant été attribuées aux variables A et B, le calculateur répondra 20.

Somme de deux nombres

Sur cet exemple, nous voulons faire la somme de 2 nombres quelconques : nous allons, en cours d'exécution du programme, demander à l'opérateur les valeurs des nombres qu'il veut additionner.

Lorsque l'instruction

```
10 INPUT "Premier nombre? ";A
```

est exécutée, le message "Premier nombre?" est affiché à l'écran, puis le calculateur attend que l'opérateur entre la valeur qu'il veut affecter à la variable A.

L'instruction 20 s'exécute dans les mêmes conditions.

L'instruction :

```
30 PRINT "Somme=";A+B
```

affiche d'abord le message 'Somme=' puis la valeur de A+B

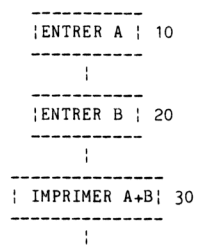
Enfin, l'instruction :

```
40 GOTO 10  (ALLER EN 10)
```

provoque un branchement à l'instruction 10 qui s'exécute à nouveau.

Tout ce qui suit REM (REMARK) où ' (apostrophe) est considéré comme commentaire.

```
LOAD "EX1"        <----------- COMMANDE pour CHARGER le PROGRAMME

OK
LIST              <---------- COMMANDE pour LISTER le PROGRAMME


2 REM          SOMME DE 2 NOMBRES
4 REM          ==================
6 '
10 INPUT "Premier  nombre? ";A         'Entre la valeur de A
20 INPUT "Deuxième nombre? ";B         'Entre la valeur de B
30 PRINT "Somme=";A+B                  'Imprime la valeur de A+B
40 GOTO 10                             'Branchement  en  10

RUN                <---------- Demande d'EXECUTION
Premier  nombre? 60
Deuxième nombre? 40
Somme= 100
Premier  nombre? 70
Deuxième nombre? 50
Somme= 120
Premier  nombre?          <---------- On appuie sur BREAK ou contrôle C pour
                                      INTERROMPRE l'EXECUTION
```



Organigramme du programme

Somme de 10 nombres

(Notion de compteur de boucles)

Pour additionner plusieurs nombres (10 par exemple), il nous faudrait, si nous procédions comme sur l'exercice précédent, écrire 10 fois la même instruction.

Utilisons plutôt une 'Boucle' et définissons une variable 'Compteur' qui compte le nombre de passages dans la boucle à l'exécution du programme.

Nous testons le passage à 10 du compteur pour 'sortir' de la boucle.

L'instruction : 90 IF COMPTEUR=10 GOTO 200 (SI COMPTEUR =10 ALLER EN 200)

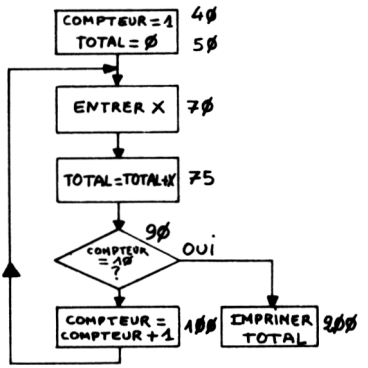
signifie que si la variable compteur devient égale à 10, le programme doit continuer en 200.

Si au contraire, le compteur n'est pas égal à 10, le programme continue à s'exécuter dans l'ordre normal des numéros d'instructions, c'est-à-dire qu'il passe à l'instruction 100.

Remarque : La commande 'TRON' (TRACE ON), généralement utilisée au moment de la mise au point, ne l'a été sur cet exemple, que pour bien montrer comment se 'déroule' un programme. La 'TRACE' est annulée par la commande 'TROFF'

```
10  '     SOMME DE 10 NOMBRES
20  '     ===================
30  '
40 COMPTEUR=1
50 TOTAL=0
60  '
70 INPUT "NOMBRE? ";X              'ENTRER NOMBRE X
75 TOTAL=TOTAL+X
80  '
90 IF COMPTEUR=10 GOTO 200         'SI COMPTEUR=10 ALLER EN 200
100 COMPTEUR=COMPTEUR+1
110 GOTO 70                        'ALLER EN 70
120  '
200 PRINT "TOTAL:";TOTAL           'IMPRIMER TOTAL
```

```
OK
run
NOMBRE? 4
NOMBRE? 6
NOMBRE? 7
NOMBRE? 5
NOMBRE? 3
NOMBRE? 5
NOMBRE? 9
NOMBRE? 8
NOMBRE? 6
NOMBRE? 5
TOTAL= 58

OK
TRON     <------ TRACE ON

OK
run
[10][20][30][40][50][60][70]NOMBRE? 4
[75][80][90][100][110][70]NOMBRE? 6
[75][80][90][100][110][70]NOMBRE? 7


[75][80][90][100][110][70]NOMBRE? 6
[75][80][90][100][110][70]NOMBRE? 5
[75][80][90][200]TOTAL: 58

OK
```



Tables

Lorsque nous avons calculé la somme de 10 nombres, nous n'avons pas gardé leur trace en mémoire centrale.

Mais si nous devons effectuer sur ces nombres plusieurs traitements (autres que leur somme), il nous faut en disposer en mémoire centrale. Nous allons les y stocker et de préférence, sous forme d'une table afin de faciliter leur traitement.

Une table est composée d'éléments auxquels on accède de la façon suivante :

- pour affecter la valeur 15 au troisième élément de la table, nous écrivons :

```
A(3)=15
```

- plus généralement, nous accédons aux différents éléments de la table A à l'aide d'un indice, I par exemple.

TABLE A

A(1)	10
A(2)	20
A(3)	15
A(4)	12

```
10 I=1
20 INPUT "Nombre? ";X
25 A(I)=X
30 IF I=5 GOTO 100
40 I=I+1
50 GOTO 20
```

La première fois I étant égal à 1, l'instruction 25 est équivalente à :

```
A(1)=X
```

Ainsi A(1) prend la valeur de X que nous avons entré dans l'instruction 20.

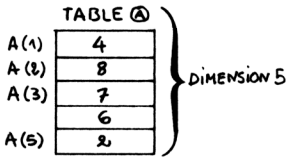
En 40 nous faisons I=I+1. I prend donc la valeur 2 et au prochain passage dans la BOUCLE l'instruction 25 sera équivalente à :

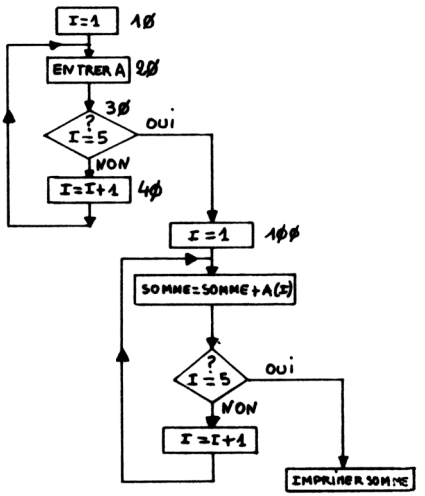
```
A(2)=X
```

etc.

Nous avons ensuite calculé la somme des éléments de cette table.

```
2 '        SOMME DES ELEMENTS D'UNE TABLE
4 '        ==============================
6 '
8 '
10 I=1
20 INPUT "NOMBRE? ";A(I)        ' Remplissage de la table A
30 IF I=5 GOTO 100
40 I=I+1:GOTO 20
45-'
100 I=1                         ' I=1 pour se positionner sur le premier
110 SOMME=SOMME+A(I)            ' nombre de la table
120 IF I=5 GOTO 300
130 I=I+1:GOTO 110
150-'
300 PRINT "SOMME=";SOMME
305 PRINT "Moyenne:";SOMME/5
310 STOP                        ' Fin du programme

OK
run

NOMBRE? 4
NOMBRE? 8
NOMBRE? 7
NOMBRE? 6
NOMBRE? 2
SOMME= 27
Moyenne: 5.4

BREAK IN 310


PRINT A(3),A(5)  <----- MODE DIRECT

 7              2

OK

PRINT A(3)+A(5)

 9

OK

PRINT SOMME/5

5.4

OK
```





Boucle 'FOR'

Le programme précédent aurait pu être écrit :

```
10 FOR I=1 TO 5              (POUR I=1 JUSQU'A 5 FAIRE)
20   INPUT "Nombre? ";X
25   A(I)=X
30 NEXT I                    (I=I+1 puis  ALLER en 20 si I<=5)
```

Cette forme d'écriture équivalente à la précédente est plus concise et facilite ainsi l'écriture et la lecture des programmes.

Toutes les instructions comprises entre FOR et NEXT sont exécutées d'abord avec la valeur I=1, puis exécutées à nouveau avec I=2 etc... jusqu'à ce que I ait une valeur égale à la limite indiquée dans l'instruction FOR. Ensuite, le programme continue normalement après l'instruction NEXT.

En 'sortant' de la boucle, I a la valeur limite indiquée +1 (6 sur l'exemple).

Autres exemples de boucle 'FOR'

Table des carrés :

```
10 FOR I=1 TO 10
20   PRINT I,(I*I)
30 NEXT I

RUN
        1       1
        2       4
        3       9
        4       16
        5       25
        .       ..
        .       ..
        10      100
```

Table de multiplications :

```
10 INPUT "Quelle Table? ";X
20 '
30    FOR I=1 TO 10
40      PRINT X;"*";I;"=";X*I
50    NEXT I
60'
70 GOTO 10

RUN
Quelle Table? 7

  7*1=7
  7*2=14
   .
   .
  7*10=70
```

Tri des éléments d'une table

Rangeons dans l'ordre croissant les éléments d'une table. Pour cela, nous allons utiliser l'algorithme suivant :

Nous comparons d'abord le deuxième élément de la table au premier :

- s'il est plus grand, nous les laissons dans l'ordre
- s'il est plus petit, nous les inversons afin qu'ils soient dans l'ordre.

```
110 IF A(I+1)<A(I) THEN    SWAP   A(I+1),A(I):INVERSION=1
      !                     !      !
   (SI A(I+1)<A(I) ALORS ECHANGER A(I+1),A(I):INVERSION=1)
```

Ensuite, nous progressons de 1 pas en comparant de la même façon le troisième élément au deuxième, et nous les inversons s'ils ne sont pas dans l'ordre.

etc.

Lorsque nous arrivons à la fin de la table, nous regardons s'il y a eu ou non inversion :

- s'il n'y a pas eu d'inversion, c'est que tous les éléments sont dans l'ordre ;
- s'il y a eu au moins une inversion, nous nous repositionnons en début de table et nous recommençons.

Après la première exploration de la table, le plus grand des éléments de la table est en fin de table.

Après la deuxième exploration de la table, le plus grand des éléments restants est en avant-dernière position.

etc.

Ainsi, après plusieurs passages dans la table (N au maximum), tous les éléments seront dans l'ordre).

Sur l'exemple, nous explorons systématiquement toute la table. En fait, nous pourrions réduire la taille de la table explorée de 1 à chaque passage.

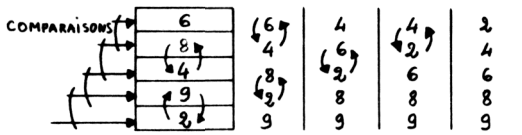
La variable inversion sert de témoin pour tester en fin de table si on a, au cours de l'exploration de la table, fait une inversion.

Si l'instruction SWAP n'existe pas, faire :

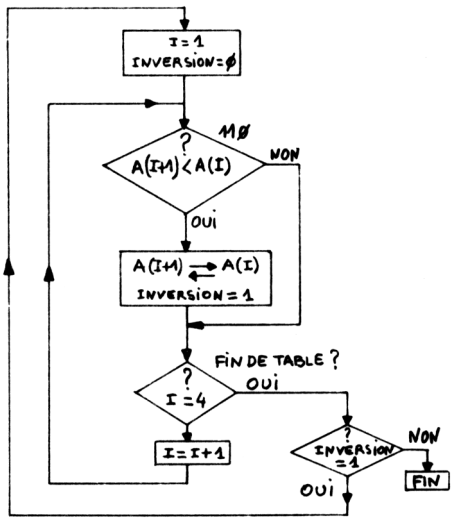
```
X=A(I):A(I)=A(I+1):A(I+1)=X
```

Remarque : la méthode de tri utilisée ('ripple sort') qui a l'avantage d'être simple à comprendre (et à programmer) est rarement utilisée en pratique dès que le nombre d'éléments à trier devient important. C'est en effet une des plus lentes que l'on puisse trouver.

```
10 REM    TRI DES  NOMBRES D'UNE TABLE
20 '      ============================
30 '
40 FOR I=1 TO 5                        '                    ACQUISITION DES NOMBRES
50   INPUT "NOMBRE? ";A(I)
60 NEXT I
70 '-----------------------------------------------------------------------
80 INVERSION=0                         '                           TRI DES NOMBRES
90 '
100 FOR I=1 TO 4
110   IF A(I+1)<A(I) THEN SWAP A(I+1),A(I):INVERSION=1
120 NEXT I
130 '
140 IF INVERSION=1 GOTO 80             ' Y a t'il eu inversion?
150 '
160 '-----------------------------------------------------------------------
170 PRINT "RESULTAT":PRINT             '              IMPRESSION DES NOMBRES TRIES
180 '
190 FOR I=1 TO 5
200   PRINT I,A(I)
210 NEXT I
```




```
run

NOMBRE? 6
NOMBRE? 8
NOMBRE? 4
NOMBRE? 9
NOMBRE? 2

RESULTAT

1          2
2          4
3          6
4          8
5          9

OK
```



Si SWAP n'existe pas ,faire:

```
X=A(I):A(I)=A(I+1):A(I+1)=X
```

Sur l'exemple précédent, nous avions prévu de trier systématiquement 5 nombres. Si nous devons trier un nombre de nombres non prévu à l'avance, nous écrivons :

```
20 INPUT "Nombre de nombres à trier ? ";NB
30 DIM A(NB)
```

L'instruction 30 réserve la place en mémoire centrale pour une table de dimension NB. Il n'était pas nécessaire sur l'exemple précédent de réserver une place pour la table dont la dimension était inférieure à 10.

Le nombre 5 qui représentait le nombre d'éléments doit être remplacé par NB. Ainsi, le 'remplissage' de la table devient :

```
FOR I=1 TO NB
  INPUT "Nombre ? ";A(I)
NEXT I
```

Tri de noms

Nous n'avons utilisé jusqu'à présent que des variables dites numériques. Or, en gestion, nous devons souvent traiter des informations non numériques telles que des noms de clients par exemple, constitués d'une succession de caractères.

Pour distinguer les variables de ce type (chaînes de caractères), il nous suffit d'ajouter au nom de la variable le caractère '$' : par exemple, la table A devient la table A$.

La comparaison de 2 chaînes de caractères s'écrit de la même façon que la comparaison de 2 nombres :

```
(IF A$(I+1)<A$(I) THEN ....)
```

Aussi le programme de tri de noms ne diffèrera-t-il de celui de tri de nombres que par l'écriture des variables.

```
10 REM    TRI DES  NOMS D UNE TABLE
20 '      ========================
30 '
32 '
34 INPUT "Nombre de NOMS a trier? ";NB
36 DIM A$(NB)
38 '
40 FOR I=1 TO NB                    '                ACQUISITION DES NOMS
50   INPUT "NOM? ";A$(I)
60 NEXT I
70 '-----------------------------------------------------------------------
80 INVERSION=0                      '                       TRI DES NOMS
90 '
100 FOR I=1 TO NB-1
110   IF A$(I+1)<A$(I) THEN SWAP A$(I+1),A$(I):INVERSION=1
120 NEXT I
130 '
140 IF INVERSION=1 GOTO 80          ' Y a t'il eu inversion?
150 '
160 '-----------------------------------------------------------------------
170 PRINT "RESULTAT":PRINT          '            IMPRESSION DES NOMS TRIES
180 '
190 FOR I=1 TO NB
200   PRINT I,A$(I)
210 NEXT I
```

```
run

Nombre de NOMS a trier? 6

Nom? DUPONT Jean
Nom? CORNU Caroline
Nom? BALU Thierry
Nom? LANGEARD Nicole
Nom? SALBREUX Catherine
Nom? GOULMOT Catherine

RESULTAT

1       BALU Thierry
2       CORNU Caroline
3       DUPONT Jean
4       GOULMOT Catherine
5       LANGEARD Nicole
6       SALBREUX Catherine
```

```
10 '                HISTOGRAMME
20 '                ===========
30 '
40 '  On génère à partir d'une table A ,une table HIST:
50 '  contenant dans la case 1 le nombre de 1 de la table A
60 '  contenant dans la case 2 le nombre de 2 de la table A
70 '
80 '  etc...
90 '
100 '         Table A              Table HIST
110 '         --------
120 '         :  3   :
130 '         --------
140 '         :  2   :
150 '         --------              -------
160 '         :  3   :            1 :  2  :
170 '         --------              -------
180 '         :  1   :            2 :  3  :
190 '         --------  ---->       -------
200 '         :  2   :  ---->     3 :  2  :
210 '         --------              -------
220 '         :  1   :            4 :  1  :
230 '
240 '
250 DIM A(20),HIST(10)
260 '                                   Remplissage de la  TABLE A
270  FOR I=1 TO 20
280    INPUT "nombre? ";X
290    IF X>10 THEN PRINT "Erreur .Nombre trop grand":GOTO 280
300    A(I)=X
310  NEXT I
320 '-----------------------------------------------------------------------
330  FOR I=1 TO 20       '              Génération  de la table HIST
340    X=A(I)
350    HIST(X)=HIST(X)+1
360  NEXT I
370 '-----------------------------------------------------------------------
380 '                                   Impression HISTOGRAMME
390  FOR I=1 TO 10
410    LPRINT I;
420 '
430    FOR J=1 TO HIST(I)
440      IF HIST(I)=0 GOTO 480
450      LPRINT "*";              ' Edition sur terminal esclave (imprimante)
460    NEXT J
470 '
480    LPRINT ""
490  NEXT I
 1 **
 2 ***
 3 **
 4 *
 5 **
 6 ***
 7 ****
 8 ***
 9
 10
```

Conseils pratiques (important)

Les langages interprétés (et c'est vraisemblablement le cas de votre BASIC) offrent des facilités pour la mise au point dont il serait dommage de ne pas profiter :

- vous pouvez interrompre votre programme en cours d'exécution en appuyant sur la touche 'BREAK' (ou contrôle C), puis visualiser les variables en MODE DIRECT afin de vous rendre compte de l'EVOLUTION de leurs valeurs en cours d'exécution.

Par exemple PRINT COMPTEUR

l'exécution du programme se poursuit en frappant la commande CONT (CONTINUE)

Les valeurs des variables peuvent être modifiées en mode direct :

```
COMPTEUR=8    'Pour affecter la valeur 8 à COMPTEUR
CONT          'Pour CONTINUER l'exécution
```

Il est permis, après une interruption, de reprendre l'exécution à une adresse quelconque X, en écrivant en mode direct :

```
GOTO X
```

Ceci évite d'avoir à exécuter à nouveau des séquences déjà testées. Les variables ne sont pas initialisées à zéro comme lors d'un RUN.

Quasiment toutes les instructions sont exécutables en mode direct, y compris les instructions de manipulation de fichiers.

A N N E X E I I

MESSAGES D'ERREURS DU BASIC MICROSOFT

Si le message d'erreur ne suffit pas pour détecter d'où provient l'erreur, on pensera à visualiser les valeurs des variables en 'mode direct'. Ceci aidera bien souvent à la retrouver.

Des instructions STOP judicieusement placées permettront également de mieux suivre l'évolution des valeurs des variables. On pourra aussi insérer momentanément des instructions de visualisation des valeurs des variables (PRINT "X=";X par exemple).

La trace (avec TRON), un peu trop riche en informations, ne sera utilisée que dans les cas particulièrement délicats.

Message	Code	
Bad file mode	54	(mauvaise utilisation de fichier) On utilise PUT,GET avec un fichier ouvert en séquentiel où on ouvre un fichier dans un autre mode que 'R' '0' ou 'I'
Bad file number	52	(mauvais numéro de fichier) On a tenté de référencer un numéro de fichier qui n'existe pas.
Bad file name	64	(mauvais nom de fichier) Le nom de fichier utilisé n'est pas normalisé (trop de caractères par exemple).
Bad record number	63	(mauvais numéro d'enregistrement) Dans un PUT ou un GET, on essaie de référencer un enregistrement de numéro > 32767 ou égal à 0.
Can't continue	17	(l'exécution ne peut se poursuivre) On a tenté de poursuivre l'exécution d'un programme qui : - a été stoppé à cause d'une erreur - a été modifié après une interruption - n'existe pas. On peut cependant continuer par GOTO xx (RUN xx initialise les variables à 0)
Disk full	61	(disque plein) Tout l'espace disque est utilisé.
Disk I/O error	57	(erreur disque) Une erreur disque s'est produite pendant une lecture/écriture. Le système d'exploitation ne peut rien faire.

Message	Code	
Division by zero	11	(division par zéro)
Field overflow	50	(dépassement du field) Une instruction FIELD a tenté d'allouer dans le buffer plus de caractères qu'il a été prévu à l'ouverture du fichier.
File already exist	58	(fichier déjà existant) Le nom de fichier défini dans NAME existe déjà.
File already open	55	(fichier déjà ouvert) On essaie d'ouvrir en mode séquentiel OUTPUT un fichier déjà ouvert ou un KILL tente de supprimer un fichier ouvert.
File not found	53	(fichier non trouvé) Une instruction LOAD,KILL ou OPEN référence un fichier qui n'existe pas.
For without next	26	(FOR sans NEXT) Un FOR sans NEXT a été détecté.
Illegal direct	12	(illégal en mode direct) L'instruction frappée n'est pas valide en mode direct, elle ne peut être exécutée que précédée d'un numéro de ligne. Ex. : instruction INPUT- DEF FN
Illegal function call	5	(appel illégal de fonction) Un paramètre hors du domaine normal a été passé à une fonction arithmétique ou une fonction chaîne. Ex. : - argument négatif pour SQR - longueur spécifiée dans LEFT$,MID$,RIGHT$ non comprise entre 0 et 255.
Input past end	62	(dépassement de fin de fichier) Une instruction INPUT#.. est exécutée alors qu'il n'y a plus d'information à lire (fichier vide éventuellement). Pour éviter cette erreur, utiliser la détection de fin de fichier EOF.
Internal error	51	(erreur interne) Une erreur système s'est produite.
Line buffer overflow	23	(dépassement du buffer de ligne) On essaie d'entrer une ligne avec trop de caractères.
Missing operand	22	(opérande manquant) Une expression contient un opérateur sans opérande.

Message	Code	
NEXT without FOR	1	(NEXT sans FOR) Une variable dans un NEXT ne correspond à aucun FOR. Ex. : La ligne FOR correspondante a été effacée sans le NEXT associé.
No resume	19	(pas d'instruction RESUME) Une routine de traitement d'erreur qui ne contient pas d'instruction RESUME a été appelée.
Out of data	4	(Data épuisés) Un READ est exécuté alors qu'il n'y a plus de DATA à lire. - On a oublié des DATA - On a oublié de programmer RESTORE
Out of memory	7	(plus de mémoire centrale) Il n'y a plus assez de place en mémoire centrale. - On doit supprimer une partie du programme ou effacer des tableaux (ERASE).
Out of string space	14	(plus de place pour les chaînes) L'espace pour les chaînes n'est pas suffisant (voir CLEAR).
Overflow	6	(dépassement de capacité) Ex. : On a tenté de donner à une variable entière une valeur en dehors de -32768,+32767.
Redimensionned array	10	(tableau redimensionné) Un tableau est à nouveau dimensionné ou une dimension de tableau est déclarée pour un tableau non déclaré explicitement mais créé par BASIC (avec une dimension 10) parce qu'il a déjà été référencé. (par une instruction A(4)=X par exemple)
Resume without error	20	(RESUME sans erreur) Une instruction RESUME a été exécutée alors qu'il n'y avait pas d'erreur.
RETURN without GOSUB	3	(RETURN sans GOSUB) - On est 'entré' dans un sous-programme par GOTO (au lieu de GOSUB). - On est 'entré' dans un sous-programme par erreur parce que l'on a oublié STOP ou END à la fin de l'exécution de son programme (devant un sous-programme)

Message	Code	
String formula too complexe	16	(expression chaîne trop complexe) Une expression du type chaîne est trop longue ou trop complexe. La décomposer en expressions plus courtes.
String too long	15	(chaîne trop longue) Une chaîne ne peut dépasser 255 caractères.
Subscript out of range	9	(référence en dehors du domaine) Un tableau est référencé en dehors de ses dimensions. Souvent, pour un tableau non déclaré qui a été dimensionné à 10 par BASIC parce que référencé (par une instruction A(4)=X par exemple).
Syntax error	2	(erreur de syntaxe) La ligne contient une erreur de syntaxe : - parenthèses non appairées - ponctuation incorrecte - instruction n'existant pas - etc.
Type mismatch	13	(désaccord entre numérique et chaîne) Une valeur numérique est affectée à une chaîne où l'inverse.
Undefined line	8	(numéro de ligne indéfini) Une instruction référence une ligne qui n'existe pas.
Undefined user fonction	18	(fonction utilisateur indéfinie) Une fonction USR est appelée avant d'être définie.
Unpritable error	21	(il n'y a pas de message pour cette erreur).

A N N E X E I I I

TABLE DES CODES ASCII

0	@c	32	SPACE	64	@	96	`
1	Ac	33	!	65	A	97	a
2	Bc	34	"	66	B	98	b
3	Cc	35	#	67	C	99	c
4	Dc	36	$	68	D	100	d
5	Ec	37	%	69	E	101	e
6	Fc	38	&	70	F	102	f
7	Gc (BEL)	39	'	71	G	103	g
8	Hc (BS)	40	(	72	H	104	h
9	Ic (HT)	41	)	73	I	105	i
10	Jc (LF)	42	*	74	J	106	j
11	Kc	43	+	75	K	107	k
12	Lc	44	,	76	L	108	l
13	Mc (CR)	45	-	77	M	109	m
14	Nc	46	.	78	N	110	n
15	Oc	47	/	79	O	111	o
16	Pc	48	0	80	P	112	p
17	Qc	49	1	81	Q	113	q
18	Rc	50	2	82	R	114	r
19	Sc	51	3	83	S	115	s
20	Tc	52	4	84	T	116	t
21	Uc	53	5	85	U	117	u
22	Vc	54	6	86	V	118	v
23	Wc	55	7	87	W	119	w
24	Xc	56	8	88	X	120	x
25	Yc	57	9	89	Y	121	y
26	Zc	58	:	90	Z	122	z
27	[c (ESC)	59	;	91	[	123	{
28	\c	60	<	92	\	124	|
29	]c	61	=	93	]	125	}
30	^c	62	>	94	^	126	~
31	_c	63	?	95	_	127	DEL

A N N E X E I V

LISTE DES PROGRAMMES

- Saisie d'un tableau à 2 dimensions 27
- Impression d'un tableau à 2 dimensions 50
- Recherche dichotomique 51
- Recherche automatique d'échelle pour histogramme 52
- Tri par la méthode de shell 53
- Création - consultation - modification d'un fichier direct 67
- Ecriture - lecture d'un fichier séquentiel 71
- Date fichier 79
- Tri bubble 80
- Tri shell-metzner 81
- Liste triée d'un fichier à accès direct 82
- Regroupement par code 83
- Allocation dynamique 85-86
- Saisie écran TRS80 90
- Saisie écran LX500 94
- Accès indexé 95
- Gestion base de données 117

A N N E X E V

INDEX ALPHABETIQUE

ABS..... 43
AND..... 19
ASC..... 31
ATN..... 43
AUTO..... 13
Bases..... 105
Bubble..... 80
CDBL..... 44
CHAIN..... 47
CHR$..... 32
CINT..... 44
CLEAR..... 13
CLOSE..... 64
COMMON..... 47
COS..... 44
CSNG..... 44
CVD..... 61
CVI..... 61
CVS..... 61
Chaînes..... 28
Commandes..... 13
DATA..... 27
DEF FN..... 45
DEFDBL..... 16
DEFINT..... 16
DEFSNG..... 16
DEFSTR..... 16
DELETE..... 13
DIM..... 25
Dichotomie..... 51
ELSE..... 21
EOF..... 75
EQV..... 19
ERASE..... 25
ERL..... 43
ERR..... 43
ERROR..... 43
EXP..... 44
Editeur..... 12
Editions..... 35
Expressions..... 18
FIELD#..... 59
FILES..... 13
FIX..... 44
FOR..... 22
FRE(0)..... 44
FRE(X$)..... 35
Fichier direct... 57
Fichier séquent.. 69
Fichier virtuel.. 87
Fonctions..... 43
GET #..... 61
GOSUB..... 40
HEX$..... 35
Histogramme..... 52
IF..... 21
IMP..... 19
INP..... 48
INPUT..... 20
INPUT#..... 70
INPUT$..... 20
INPUT$#..... 75
INSTR..... 33
INT..... 44
Indexé..... 33
KILL..... 13
LEFT$..... 29
LEN..... 31
LINE INPUT..... 20
LINE INPUT#..... 72
LIST..... 13
LLIST..... 14
LOAD..... 14
LOC..... 64
LOF..... 64
LOG..... 44
LPOS..... 39
LPRINT..... 39
LSET..... 60
MERGE..... 14
MID$..... 29
MKD$..... 61
MKI$..... 61
MKS$..... 61
MOD..... 18
NAME..... 14
NEW..... 14
NEXT..... 22
NOT..... 19
OCT$..... 34
ON ERROR..... 43
ON GOSUB..... 42
ON GOTO..... 41
OPEN..... 58
OPEN..... 69
OR..... 19
OUT..... 48
Opérateurs..... 18
PEEK..... 48
POKE..... 48
POS..... 39
PRINT..... 35
PRINT USING..... 36
PRINT#..... 70
PUT#..... 60
RANDOMIZE..... 44
READ..... 27
RENUM..... 14
RESTORE..... 27
RESUME..... 43
RETURN..... 40
RIGHT$..... 29
RND..... 45
RSET..... 60
RUN..... 14
Ripple..... 82
SAVE..... 14
SGN..... 45
SIN..... 45
SPACE$..... 34
SPC..... 36
SQR..... 44
STEP..... 22
STR$..... 31
STRING$..... 34
SWAP..... 46
SYSTEM..... 14
Saisie écran..... 88
Shell..... 53
Shell-Metzner.... 81
Sous-programmes.. 40
TAB..... 36
TAN..... 45
THEN..... 21
TRI..... 53
TROFF..... 15
TRON..... 15
Tableaux..... 25
VAL..... 31
Variables..... 15
WAIT..... 48
WEND..... 46
WHILE..... 46
WIDTH..... 39
WRITE..... 39
WRITE#..... 73
XOR..... 19

Achevé d'imprimer
sur les presses de
Impressions Cepi
174, Quai de Jemmapes
75010 PARIS
Dépot légal N ° 58 - 2e trimestre 1980

N ° d'édition : 86470-10-7
ISBN : 2-86470-010-7

P.S.I.

* *

Le BASIC et ses Fichiers

Enfin un « Basic » qui s'intéresse sérieusement (attention : série rouge) à la programmation des applications utilisant des Fichiers sur disquette ou sur disque. Ce livre contient de nombreux exemples de programmes commentés. La version de Basic retenue, car il a fallu choisir, est la 5. de Microsoft fonctionnant sous CP/M. C'est dire que cet ouvrage intéresse directement les utilisateurs de PSI à microprocesseur Z.80.

Editions du P.S.I.
9, rue d'Orgemont
77400 Lagny/Marne

ISBN : 2-86470-010-7

Imprimé en France

Jacques Boisgontier

P.S.I.

Le BASIC et ses Fichiers